2

Reinforcement Learning for the Adaptive Control of Perception and Action

Steven D. Whitehead

Technical Report 406
February 1992

UNIVERSITY OF
ROCHESTER
COMPUTER SCIENCE

Reinforcement Learning for the Adaptive Control of Perception and Action

by

Steven Douglas Whitehead

Submitted in Partial Fulfillment

of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

Supervised by Dana H. Ballard

Department of Computer Science

University of Rochester

Rochester, New York

February 1992

Curriculum Vitae

Steven Douglas Whitehead was born in Seattle Washington on October 30, 1960. He graduated from Shoreline High School in 1978. In January of 1982, he was inducted into the Phi Beta Kappa (Arts and Sciences) Honor Society, and in June of 1982, he received a B. S. in Physics, *cum laude*, from Washington State University, Pullman, Washington.

Upon graduation, he accepted an Industrial Undergraduate Research Fellowship from GTE Laboratories, Waltham, Massachusetts, where he studied the spectral properties of solid state lasers. While at GTE, Steven came to know John N. Daigle and through discussions with him became interested in computers and communications. Starting in September, 1982, he attended Clemson University, Clemson, South Carolina, where he studied computer communications and queueing theory under the direction of Dr. Daigle. In December, 1984, he completed his masters thesis, entitled "An Alternative Approach to Queueing Systems Via Generalized Balance Equations," and received an M. S. in Electrical Engineering.

Steven then moved to Mountain View, California, where he joined the Artificial Intelligence Group of GTE Government Systems. For the next two years he participated in the development of knowledge based systems for government intelligence and logistics support applications.

In September 1986, he entered the Doctoral Program of Computer Science at the University of Rochester. His work at Rochester included two years of teaching and research duties. He also served as an Undergraduate Advisor and as a member of the Graduate Admissions Committee. During the 1989-90 academic year, he served as the Graduate Student Representative to the faculty. His thesis work has been the result of an enjoyable collaboration with Dana H. Ballard.

Acknowledgments

The work described herein is the result of a close collaboration with Dana Ballard, my advisor and benefactor. I thank him for his steadfast encouragement and advice, for his unwavering enthusiasm, and for all the fun we have shared over the years. I would also like to thank my committee, James Allen, Mark Fulk, Henry Kyburg, and Rich Sutton for their help and tutelage. I am especially indebted to Rich for guiding me through the reinforcement learning literature. In addition, this work has been improved through discussions with Phil Agre, Andy Barto, Chris Brown, David Chapman, and Leslie Kaelbling, Josh Tenenburg, Bulent Murtezaoglu, Randall Nelson, and Lambert Wixson.

Special thanks also go to Chris Brown and Kathy Ivey for detailed comments on a draft of this document.

The people in my department — the faculty, staff, and the students — make this department unique, and I am thankful to them for making this a wonderful and cooperative place to work. Thanks especially to Mike Swain, Paul Cooper, Ray Rimey, David Coombs, Leo Hartman, Hans Koomen, Nat Martin, Ray Frank, and Jay and Susan Weber for many fascinating and illuminating conversations. Thanks to Peg, Pat, Jill, Peggy, Luid, Brad, Jim, and Tim, who were always willing to guide me through the administrative and technical obstacles.

Finally, this work is dedicated to my family. To Binh, Anna Marie, and the cats, for making life at home even more fulfilling than life at school, and to my parents for unflagging support and infinite patience.

*This material is based upon work supported by the National Science Foundation under Grant number IRI-8903582 and in part by ONR research contract no. N00014-80-C-0197.

Abstract

This dissertation applies reinforcement learning to the adaptive control of active sensory-motor systems. Active sensory-motor systems, in addition to providing for overt action, also support active, selective sensing of the environment. The principal advantage of this active approach to perception is that the agent's internal representation can be made highly task specific — thus, avoiding wasteful sensory processing and the representation of irrelevant information. One unavoidable consequence of active perception is that improper control can lead to internal states that confound functionally distinct states in the external world. This phenomenon, called *perceptual aliasing*, is shown to destabilize existing reinforcement learning algorithms with respect to optimal control.

To overcome these difficulties, an approach to adaptive control, called the *Consistent Representation* (CR) method, is developed. This method is used to construct systems that learn not only the overt actions needed to solve a task, but also where to focus their attention in order to collect necessary sensory information. The principle of the CR-method is to separate control into two stages: an identification stage, followed by an overt stage. The identification stage generates the task-specific internal representation that is used by the overt control stage. Adaptive identification is accomplished by a technique that involves the detection and suppression of perceptually aliased internal states. Q-learning is used for adaptive overt control.

The technique is then extended to include two cooperative learning mechanisms, called *Learning with an External Critic* (LEC) and *Learning By Watching* (LBW), respectively, which significantly improve learning. Cooperative mechanisms exploit the presence of helpful agents in the environment to supply auxillary sources of trial-and-error experience and to decrease the latency between the execution and evaluation of an action.

Table of Contents

Curriculum Vitae ii

Acknowledgments iii

Abstract iv

List of Tables vii

List of Figures viii

1 Introduction 1

1.1 Active Perception 2

1.2 Reinforcement Learning 3

1.3 Principal Contributions 7

1.4 Thesis Outline 9

2 Background: Active Perception 11

2.1 Principles of Active Perception 11

2.2 Visual Routines 13

2.3 Deictic Representations 22

3 Background: Reinforcement Learning 27

3.1 Markov Decision Processes 27

3.2 Dynamic Programming 31

3.3 Q-Learning 40

4 The Block Stacking Testbed **48**

4.1 The Block Manipulation Task 49

4.2 The Sensory-Motor System 52

4.3 Perceptual Mappings 57

4.4 A Formal Model of Embedded Learning 61

5 Combining Active Perception and Q-Learning **66**

5.1 Meliora-I 67

5.2 Definitions and Nomenclature 70

5.3 The Effects of Perceptual Aliasing 71

5.4 A Simple Example 73

5.5 Looking Back at Meliora-I 78

5.6 The Long Arm of Perceptual Aliasing 80

6 Learning Consistent Representations **83**

6.1 Meliora-II 84

6.2 Performance Results 89

6.3 The Consistent Representation Method 96

7 Cooperative Mechanisms **106**

7.1 Introduction 107

7.2 Learning with an External Critic 108

7.3 Learning By Watching 114

7.4 Analysis 121

8 Limitations and Future Work **134**

8.1 Separation of Perceptual and Overt Control 135

8.2 State Space Size 142

8.3 Other Areas for Future Research 145

9 Conclusions **147**

Bibliography **151**

A Proofs for Search Analysis Theorems **162**

List of Tables

5.1 The sampled utility and action-value functions and the estimated utility and action-value functions. 77

List of Figures

1.1 Task-dependent eye movements of a subject observing I. E. Repin's painting, "An Unexpected Visitor," as recorded by A. L. Yarbus. . 4

2.1 The processing stages of Ullman's visual routines model. 15
2.2 Examples of the inside-outside task. 18
2.3 Determining whether or not two X's lie on the same curve. 19
2.4 The subfigure identification task. 20
2.5 A Pengo game in progress. 26

3.1 The navigation task, an example of a sequential control problem. . 32
3.2 A formalization of the navigation task as a Markov decision process. 33
3.3 The optimal policy and the optimal action-value function for the navigation task. 34
3.4 The Policy Iteration Algorithm 38
3.5 The Value Iteration Algorithm. 40
3.6 A simple version of the 1-step Q-learning algorithm. 46

4.1 A sketch of the block manipulation task. 50
4.2 The specification for the robot's sensory-motor system. 53
4.3 A fixed set of rules that reliably solve the GB-task. 56
4.4 An example of passive abstraction in the GB-task. 58
4.5 An example of perceptual aliasing in the GB-task. 59
4.6 The relationship between states in the external and internal decision problems. 60
4.7 A formal model for an agent with an embedded learning system and an active sensory-motor system. 62

5.1 Average solution time per trial for Meliora-I on runs of 1000 trials. 69

5.2 An illustration of the difficulties caused by perceptual aliasing. . . 74
5.3 Plots of the utility versus time as the agent traverses an optimal trajectory. 75
5.4 An example of perceptual aliasing from the GB-task. 79
5.5 Inconsistent internal states act as *attractors* leading to non-optimal decisions in 1-step Q-learning. 81

6.1 The lion algorithm used by Meliora-II to learn the GB-task. . . . 85
6.2 A sketch of the Meliora-II's decision system in action. 87
6.3 A plot of the number of overt steps per trial for Meliora-II on the GB-task. 90
6.4 A plot of the average number of overt steps per trial for Meliora-II on the GB-task. 91
6.5 Plots of the average number of steps per trial for easy, intermediate, difficult, and very difficult problem instances. 93
6.6 Success rates for each of the four classes of problem instances. . . 94
6.7 The fraction of overestimations encountered over 200 runs for each of the four classes of problem instances. 95
6.8 Performance plots for experiments that include piles of up to 20 outlying blocks. 97
6.9 The basic architecture of a system using the CR-method. 98
6.10 Representing an identification procedure with a decision tree . . . 101
6.11 Detecting inconsistent decisions by partitioning the equivalence class. 102

7.1 The BB-LEC algorithm used by Meliora-III-LEC to learn the GB-task. 110
7.2 Average solution time versus trial number for Meliora-III-LEC. . . 113
7.3 The average number of suppressed decisions per trial of Meliora-III-LEC. 113
7.4 Performance of Meliora-III-LEC in response to a change in the underlying decision problem. 115
7.5 The conceptual organization of an agent using LBW. 116
7.6 The LBW algorithm used in the performance mode. 118
7.7 The LBW algorithm used in the watching mode. 119
7.8 Performance curves for Meliora-III-LBW. 122
7.9 Search time complexity as a function of state space depth. 126
7.10 A 1-dimensional homogeneous problem solving task. 129

7.11 Average solution time plots for Q0 and Q+. 130

8.1 A task that the CR-method cannot perform optimally. 136
8.2 A navigation task that requires memory. 140

1 Introduction

For the better part of thirty years research in AI has focused on high level, cognitive aspects of intelligence. Topics like planning, problem solving, natural language understanding, knowledge representation, and reasoning have traditionally been at the core of AI. However, abstract thought is only a part of intelligence and is generally useless without sensors and effectors to ground it in the real world. For these reasons an increasing number of researchers have begun to look at some of the more mundane aspects of intelligent behavior that can be modeled with complete behaving systems. More and more researchers are building robots that act in the real world instead of complex systems that act in artificial, simulated, or disembodied worlds. This shift has had two important effects. First, it has forced the reexamination of basic assumptions that underlie traditional approaches to intelligent behavior, and, second, it has led to research in areas that address the shortcomings of traditional approaches.

Two topics that are currently generating a great deal of interest are *active perception* and *reinforcement learning*. Active perception is concerned with development of sensory systems that are dynamically controlled to selectively process and represent precepts about the environment in a task-dependent way. Reinforcement learning is concerned with the adaptive control of an agent through the use of scalar rewards (for feedback) and direct trial-and-error interaction with the environment.

Active perception and reinforcement learning are both important to the development of intelligent agents. Active perception is needed for efficient, realistic perception, and reinforcement learning is important to the development of adaptive systems that, among other things, do not rely too heavily upon *a priori* domain knowledge. To date, these two lines of research have, for the most part, progressed independently.[1] Work on active perception has focused primarily on

[1]Notable exceptions include the work described in this thesis [Whitehead and Ballard, 1990; Whitehead and Ballard, 1991a], work by Tan on learning cost sensitive internal representations [Tan, 1991b; Tan, 1991a], Chapman and Kaelbling's work on the generalization problem [Chapman and Kaelbling, 1991], and Schmidhuber's work on learning to control visual attention

understanding the benefits of active vision, on developing approaches to active vision, and on building systems that use it. Little concern has been focused on how an agent might actually learn to control an active visual system. Similarly, most work in reinforcement learning has ignored perceptual issues completely by assuming that the agent at each time step has sensory inputs that completely describe the state of the world with respect to the task.

1.1 Active Perception

The vast majority of work in AI has not dealt realistically with perception. Typically, perception is abstracted out of consideration in order to focus on more central decision making issues. It is common to assume that a decoupled (often implicit) sensory system provides the decision system with an internal representation that completely and accurately describes the state of the external world. This representation frequently takes the form of a set of propositions that describe the relationships between, and the features of, every potentially relevant object in the domain. Even for simple toy domains this reconstructive approach to perception places an unrealistic burden on the sensory system and leads to internal representations rife with irrelevant information. For example, in a real world version of the blocks-world it is unrealistic to expect a sensory system to analyze more than a few blocks at a time. Moreover, if there are n blocks in the world and each is represented using traditional methods, then the size of the state space is $O(n!)$ [Ginsberg, 1989]. For $n = 20$ the state space has over forty billion $(42,949,672,940)$ states.

The large amount of information encoded in these representations is difficult to deal with, but more importantly, most of it is irrelevant to the immediate task facing the agent. It only interferes with decision making (and learning) by clogging the system with irrelevant detail. The situation deteriorates even further when we consider agents whose tasks are numerous, complex, and not well understood ahead of time. Under these circumstances, complete internal representations will necessarily have to encode even more information that is likely to be even less useful at any given point in time. If intelligent robots are to be achieved, then intelligent sensing strategies that balance generality and flexibility with computational feasibility must be developed.

Active perception represents a promising approach to this challenge. The central tenet of active perception is that an agent's sensory system is an information collecting resource that is at least partially under the control of the agent's internal decision processes. By directly controlling the allocation of sensory processing resources, the agent selectively monitors and represents those aspects of the world

[Schmidhuber, 1990a].

that are immediately relevant to the task at hand and ignores what is irrelevant. A key assumption is that at any given time only a relatively small amount of information is needed for decision making. By exploiting this assumption, the amount of comprtation required for sensing reflects the complexity of the agent's task and not the complexity of the world in which it is embedded. Efficiency is attained by carefully controlling the selection and application of computational resources so that only relevant aspects of the environment are processed, thus generating internal representations that are minimal in size and task-specific. Flexibility and generality are attained by providing the system with a range of sensory processing resources that can be flexibly applied to different parts of the environment. Additional flexibility is gained when processing resources define primitive operations that can be composed to define complex sensing routines [Ullman, 1984].

Human vision is a perfect example of active perception. We move our eyes to allocate our visual processing resources (e.g., foveal vision) on those aspects of the world that are most important to us. This point has been elegantly demonstrated by Yarbus [Yarbus, 1967], who showed that a subject's eye movements are task-dependent (i.e., see Figure 1). Yarbus' seminal experiments on human eye movements have been followed by considerable research in psychology and artificial intelligence (not to mention physiology, anatomy, and neuroscience) aimed at better understanding selective vision in man and machine. Some of this research includes work analyzing the computational advantages of active vision [Aloimonos *et al.*, 1987; Ballard and Ozcandarli, 1988; Ballard, 1989a; Ballard, 1991; Simmons, 1990; Tsotsos, 1987], work on architectures for vision and visual sensing strategies [Agre, 1988; Bajcsy and Allen, 1984; Chapman, 1990b; Chapman and Kaelbling, 1991; Chrisman and Simmons, 1991; Dickmanns, 1989; Garvey, 1976; Rimey and Brown, 1990; Swain, 1990; Tan and Schlimmer, 1990; Ullman, 1984; Romanycia, 1987; Romanycia, 1988] and aspects of active vision in humans [Chapman, 1990a; Noton, 1970; Noton and Stark, 1971a; Noton and Stark, 1971b; O'Regan and Levy-Schoen, 1983; Treismann and Gelade, 1980; Ullman, 1984].

1.2 Reinforcement Learning

An assumption that is almost universal in AI is that the agent has an *a priori* domain model which it uses to reason about possible courses of action. These models are essential to traditional planning approaches since they are required for search and plan generation. In most cases, the model takes the form of a set of individual operator models [Fikes and Nilsson, 1971; Laird *et al.*, 1986] or a set of frame axioms [Hayes, 1973; McCarthy, 1977] that are used to predict the effects of actions. It ᛬ also common to assume that the model is complete,

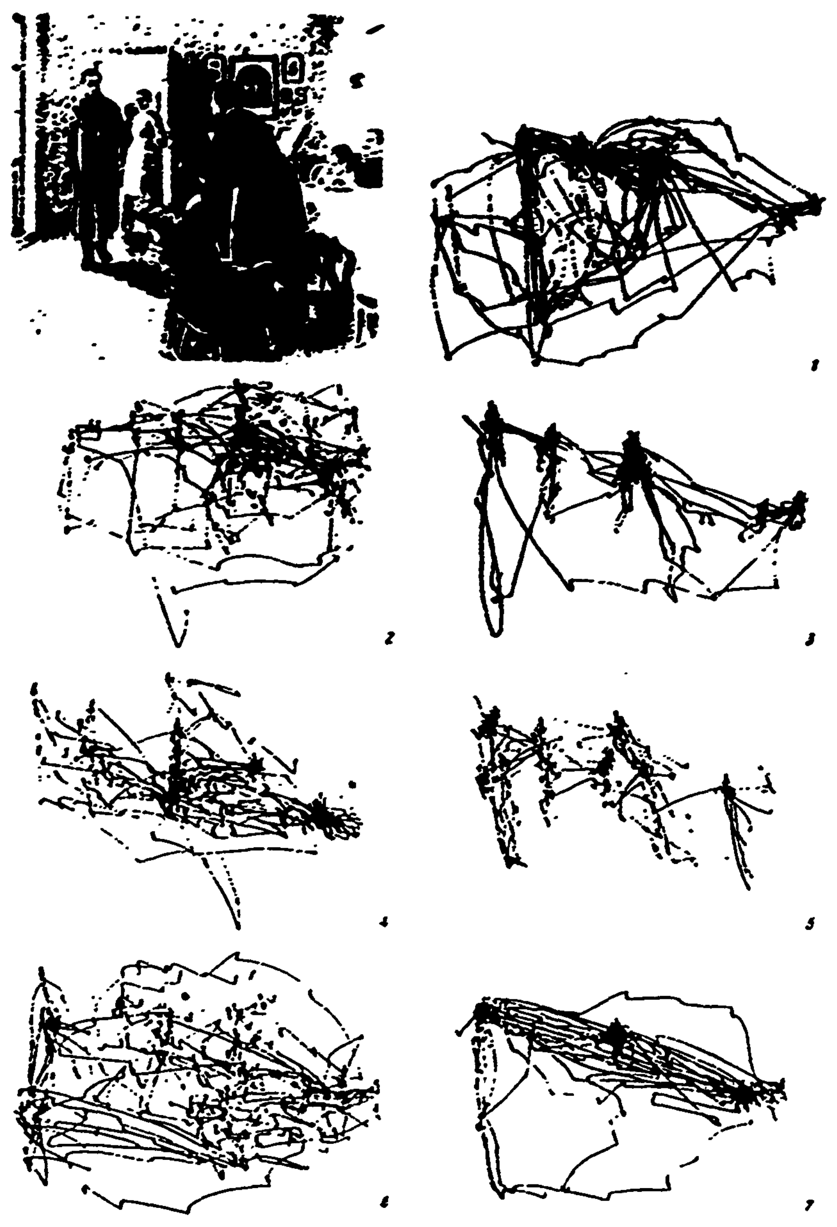


Figure 1.1: (reproduced from [Yarbus 1967]) A reproduction of I. E. Repin's painting "An Unexpected Visitor" and records of seven eye movement traces for the same subject. Each record lasted 3 minutes. The subject examined the reproduction with both eyes. 1) Free examination of the picture. Before the subsequent recording sessions, the subject was asked to: 2) estimate the material circumstances of the family in the picture; 3) give the ages of the people; 4) surmise what the family had been doing before the arrival of the "unexpected visitor"; 5) remember the clothes worn by the people; 6) remember the position of the people and objects in the room; 7) estimate how long the "unexpected visitor" had been away from the family.

accurate and stationary. Unfortunately for real-world tasks, models that satisfy these assumptions are hard to come by for the following reasons:

1. The task domain may not be well enough understood to formulate an accurate *a priori* model. In some cases it may be that the task is too complex or the domain too unconstrained. In other cases, it may be that the task to be performed cannot be anticipated in advance (e.g., as might be the case for a general purpose robot that gets trained off site or whose tasks change over time).

2. Even when the task domain is well known, it is often difficult to formulate a model that is accurate and complete. This is especially true of real-world tasks [Shafer, 1990] and when using symbolic models [McCarthy and Hayes, 1969; Hayes, 1973].

3. Most classical planning techniques depend on the world being deterministic. This assumption makes it difficult to apply classical techniques to problems that are inherently stochastic and makes classical planning unlikely for the real world, where unexpected events are commonplace.[2]

4. The real world is constantly changing. Tools wear out, parts break, objects get moved. The real world is non-stationary, and fixed *a priori* models of it are going to be inadequate. An agent's model must be adaptable so that it can capture these changes. Most symbolic models are far too fragile and inexpressive to afford incremental adaptation.

Computational models of intelligent control must not depend too heavily upon complete and accurate *a priori* domain models. If control depends upon an explicit model at all, it must suffice for it to be incomplete and inaccurate. If domain knowledge is available, then the agent should be capable of exploiting it, but it should not be a prerequisite for intelligent control.

Reinforcement learning offers an alternative approach to control that does not depend upon explicit, *a priori* domain models [Minsky, 1954; Michie and Chambers, 1968; Barto *et al.*, 1983; Holland *et al.*, 1986; Sutton, 1988; Watkins, 1989]. A reinforcement learning system is *any system that through direct interaction with its environment improves its performance by receiving feedback in the form of a scalar reward (or penalty) that is commensurate with the appropriateness of its response.* By *improves its performance* we mean that the agent uses the feedback to adapt its behavior in an effort to maximize some measure of the reward it receives in the future. Intuitively, a reinforcement learning system can be viewed as

[2]Recently, interest in probabilistic reasoning and planning has been on the rise (e.g., see [Pearl, 1988, Dean and Wellman, 1991].) Unfortunately, the shift towards probabilistic models seems only to have increased the computational complexity of classical planning methods.

a hedonistic automaton whose sole objective is to maximize the positive (reward) and minimize the negative (penalty).

The principles of reinforcement learning are pertinent to intelligent control since they lead to systems that do not depend upon *a priori* knowledge for decision making. Instead of using an explicit domain model to generate a sequence of actions (a plan) which is then executed open loop as in classical planning, a reinforcement learning system maintains an explicit policy function (analogous to a universal plan [Schoppers, 1989b; Schoppers, 1989a]) that maps situations directly into actions. In this case, at each point in time decision making reduces to computing (looking up) the value of the policy function for the current situation. If the agent's policy is correct, then its performance is optimal. If it is not, then the policy is incrementally improved through direct experience with the world. By relying on the world directly for feedback (reinforcement) these systems avoid many of the pitfalls associated with model-based control methods.[3]

Agents based on reinforcement learning confer a number of other advantages as well. First, reinforcement learning systems are both situated and reactive: they can respond quickly to unexpected contingencies and opportunities [Agre, 1988]. There is some confusion in the literature about the distinction between situated and reactive. Situatedness and reactiveness are two distinct properties. Both are required for intelligent behavior. An agent is situated if its control decision is based on the immediate situation (as determined by sensor readings and possibly a limited amount of internal state); an agent is reactive if it generates actions/behavior at a rate that is commensurate with the dynamics of the environment in which it is embedded. Reinforcement learning systems are situated since decision making is usually based on the immediate situation. They are reactive since decision making consists of evaluating a policy function, which typically requires a small constant amount of time. Second, since reinforcement learning systems incrementally adapt their policies based on experience accumulated over time, they are effective for control tasks that are stochastic and (under appropriate conditions) non-stationary. Also, reinforcement learning systems can exploit domain knowledge when it is available. This can be achieved by 1) using *a priori* knowledge to determine a good initial policy [Franklin, 1988], 2) using a domain model to perform hypothetical experiments instead of relying solely on trial-and-error experiments in the world [Whitehead, 1989;

[3]A fundamental assumption implicit in reinforcement learning is that an agent over the course of its lifetime is presented with the same set of problems over and over [Agre, 1985]. The solutions to these problems are encoded directly in the policy function. Once a solution to a particular problem is learned (encoded in the policy), future occurrences of the problem can be solved by simply following the instructions encoded in the policy — no planning or reasoning is required. A new problem (i.e., one which cannot be reduced to a previously learned problem) or a change in an existing problem will initially lead to degraded performance because the policy will not encode a solution. In this case reasoning/problem solving may be used, or the system can learn the task by trial-and-error experimentation in the environment.

Sutton, 1990a; Lin, 1990], and 3) using a model to generalize the results of experiments for better credit assignments [Yee *et al.*, 1990]. Also, because learning is incremental, these models need not be complete or accurate. This robustness in the face of incomplete models has led to systems that profit by using models which themselves have been learned [Sutton, 1990a; Sutton, 1990b; Lin, 1990].

The idea of using rewards and penalties as feedback for adaptive systems dates back at least to the late fifties and Marvin Minsky [Minsky, 1954], who studied automata that learned to solve a series of sequential decision problems (maze problems) by adjusting their decision rules based upon the receipt of rewards and penalties. Since that time, reinforcement learning has been studied widely and from a variety of perspectives. Some highlights include Minsky's early maze learning automata [Minsky, 1954]; Samuel's checker player [Samuel, 1963]; Michie and Chambers' 'boxes' algorithm [Michie and Chambers, 1968]; Holland's classifier systems and the bucket brigade algorithm [Holland and Reitman, 1978; Holland *et al.*, 1986]; Sutton's Adaptive Heuristic Critic [Sutton, 1984; Barto *et al.*, 1983], its neural implementation [Anderson, 1986], and, subsequently, Sutton's Theory of Temporal Difference Methods [Sutton, 1988]. More recently, the relationship between reinforcement learning and dynamic programming has been established [Watkins, 1989; Werbos, 1987] and a mathematical theory of reinforcement learning is beginning to emerge (e.g., see [Barto *et al.*, 1991]). Other recent results address issues such as modularity [Booker, 1988; Riolo, 1988; Mahadevan and Connell, 1991; Singh, 1991; Wixson, 1991], integration of planning, action, and learning [Whitehead and Ballard, 1989a; Whitehead and Ballard, 1989b; Whitehead, 1989; Sutton, 1990a; Sutton, 1990b; Sutton, 1991; Lin, 1990], faster credit assignment [Yee *et al.*, 1990; Lin, 1991; Whitehead, 1991], statistical foundations for better exploration strategies [Kaelbling, 1990], selective perception and generalization [Whitehead and Ballard, 1991a; Chapman and Kaelbling, 1991; Tan, 1991a], and neural implementations [Williams, 1987; Schmidhuber, 1990b].

1.3 Principal Contributions

This dissertation examines architectures that combine both active sensory systems (for feasible, task-dependent perception) and reinforcement learning (for adaptive, non-model-based control). It is shown that incorporating active perception and reinforcement learning into a single system is non-trivial because of subtle interactions that prevent the system from learning an adequate control policy. What makes learning in this context difficult is that, in addition to learning the overt actions needed to solve a problem, the agent must also discover how to control its sensory system (e.g., focus its attention) n order to represent accurately the state of the world with respect to the task. If the agent selectively attends to the few key objects relevant to the task, then its internal state accurately represents the world. If, however, the agent does not attend to those key objects, the internal state may say nothing useful about the world. A dilemma arises: in order for the agent to learn to solve a task, it must accurately represent the world with respect to the task; but, in order for the agent to learn an accurate representation, it must in some sense know how to solve the task.

The difficulty arises when the sensory system, due to improper control, generates an internal state that represents two or more functionally different situations in the external world. These internal states are said to be *inconsistent* and this undesirable overloading of internal states is called *perceptual aliasing.* Perceptual aliasing is shown to interfere severely with most existing reinforcement learning algorithms by making it impossible, in certain situations, to estimate accurately the utility of performing an action. Perceptual aliasing is a fundamental obstacle to adaptive intelligent control since it not only arises in active perception but is also inherent in virtually every abstraction and generalization mechanism — that is, perceptual aliasing can occur any time it is possible to ignore information that is relevant to decision making and utility estimation.

To surmount the problems caused by perceptual aliasing an adaptive control technique, called the *Consistent Representation* (CR) method, is developed. In the CR-method control is accomplished in two distinct phases: a state identification phase, followed by an overt control phase. During state identification, the system executes sensory-control actions in an attempt to generate an internal representation that accurately identifies the state of the external world with respect to the task. Next, during overt control, this internal representation is used to generate the overt actions needed to perform the task. Both the state identification and the overt control stages are adaptive. Learning in the overt control stage is based primarily on standard reinforcement learning techniques — that is, an overt control policy is adjusted to maximize expected future rewards. Learning for state identification is somewhat different. The objective of state identification is to generate internal states that are somehow adequate. In the CR-method, adequacy is defined in terms of whether or not an internal state is *consistent.* By detecting internal states that are inconsistent, the identification procedure used by the sensory controller can be adapted and inconsistent states can be eliminated. When this is achieved the sensory controller has learned to generate a task-dependent internal representation. In the CR-method, perceptual and overt control is learned incrementally and simultaneously. This follows since the overt control policy cannot be completely learned until some of a consistent internal representation is well established, while perceptual control cannot be accomplished until the overt controller learns, at least partially, how to solve the task. This interaction results in systems that first learn to represent and solve easy instances of a task (e.g., the last few steps of a task), and then, through a bootstrapping process, learn to solve more and more difficult instances. The

CR-method is demonstrated on a robot that learns a simple block manipulation task that requires the control of an active sensory-motor system.

Reinforcement learning systems are plagued by unstructured initial search. In particular, when rewards and punishments are rare, an agent may execute a long sequence of actions before it receives feedback essential for learning. If the agent knows little or nothing about the task *a priori*, then during the initial phases of learning lack of feedback can lead to long random walks in search of rewards. As the size and complexity of the state space is scaled these random walks quickly become prohibitive. We define cooperative mechanisms that help reduce search by providing the agent with shorter latency feedback and auxiliary sources of experience. The principal motivation for cooperative mechanisms is that, in nature, intelligent agents do not exist in isolation, but are embedded in a benevolent society that guides and structures learning. Humans learn by watching others, by being told, and by receiving criticism and encouragement. *Learning more often involves knowledge transfer than discovery.* Similarly, intelligent robots cannot be expected to learn complex tasks in isolation by trial-and-error alone. Instead, they must be embedded in cooperative environments, and algorithms must be developed to facilitate the transfer of knowledge among them.

In this work, two cooperative learning techniques are proposed and demonstrated. The first, called *Learning with an External Critic* (LEC), is based on the idea of a mentor, who watches the agent and generates immediate rewards in response to the agent's most recent actions. This reward is used to bias temporarily the agent's control strategy. The second algorithm, called *Learning By Watching* (LBW) is based on the idea that an agent can gain valuable experiences vicariously by relating the observed experiences of others to its own. These two algorithms are demonstrated in the block stacking domain and shown to improve the learning rate substantially. Also, the search time complexity for these algorithms, along with a popular reinforcement learning algorithm, is analyzed for a restricted (but representative) set of learning tasks. The results indicate that under certain circumstances a popular algorithm (Q-learning) can be expected to require time at least exponential in the size of the state space, while the LEC and LBW algorithms require time at most linear in the size of the state space, and under appropriate conditions may only require time proportional to the length of the optimal solution path. While these analytic results apply only to a restricted class of tasks, they shed light on the complexity of search in reinforcement learning in general and the value of these cooperative mechanisms for reducing search.

1.4 Thesis Outline

The technical results described in this dissertation were obtained by analyzing a series of systems developed to solve tasks in a simple simulated blocks world domain. Even though this domain is quite simple, these systems proved useful for developing my intuition. For this reason, the dissertation chronicles the evolution of these systems, collectively called Meliora. I begin by describing the first system built, which took a most straightforward approach and failed miserably; then, after analyzing its failure, I describe a system based on the CR-method, which succeeds in learning the task but is slow; following that, I demonstrate the potential of cooperative mechanisms. Grounding the discussion in the blocks world makes exposition easier and an intuitive understanding more likely. At various points in the discussion the results are generalized in a more formal theory.

The remainder of the dissertation is organized as follows. Chapters 2 and 3 review background material from the areas of active perception and reinforcement learning, respectively. The emphases in these chapters are on visual routines and deictic sensory-motor systems (for active vision) and Q-learning (for reinforcement learning) since these are the techniques used by the block stacking systems. Readers familiar with active perception, Markov decision processes, and Q-learning may wish to quickly skim Chapters 2 and 3 or skip directly to Chapter 4. Chapter 4 describes the block stacking task. This discussion includes a description of the deictic sensory-motor used by all the robots. A formal model for describing agents that integrate active perception and reinforcement learning is also developed. Chapter 5 describes experiences with the first block stacking agent, analyzes its failure and formalizes the interactions caused by perceptual aliasing. Chapter 6 describes a specific algorithm developed to overcome the effects of perceptual aliasing for the block stacking task. Generalizing this algorithm leads to the CR-method. In Chapter 7 we focus on cooperative mechanisms for improving the learning rate. This chapter describes the principles of LEC and LBW, demonstrates them in several block stacking systems, and presents a formal analysis of the search time complexity of these algorithms. Chapter 8 discusses the limitations of the CR-method and identifies areas for future research. Conclusions are drawn in Chapter 9.

2 Background: Active Perception

This chapter and the next review essential background material needed to motivate and understand the work described in this dissertation. This chapter focuses on active perception and Chapter 3 focuses on reinforcement learning. These chapters are not intended to be thorough reviews of research in active perception or reinforcement learning. Rather, the emphasis is on reviewing specific ideas that form the conceptual foundation on which our work is based. With respect to active perception, we focus on Ullman's Visual Routines model of human intermediate level vision [Ullman, 1984] and on Agre and Chapman's theory of deictic representations [Agre and Chapman, 1987; Agre, 1988; Chapman, 1990b]. With respect to reinforcement learning, we focus on Watkins' Q-learning algorithm [Watkins, 1989].

2.1 Principles of Active Perception

The purpose of a sensory system should be to provide the agent's internal decision processes with enough information to make effective control decisions. Whether or not a robot's sensors provide it with adequate information depends upon the task being performed and the capabilities of the robot's internal decision system. In any case, an important decision in the design of a robot is the choice of the aspects of the environment to sense at any given point in time. The most common approach taken in AI is to adopt a fixed sensory system that continually monitors *all* potentially relevant features of the environment. In classical planning, this approach is implicit in the objective representations that at each point in time completely describe the state of the world (e.g., [Fikes *et al.*, 1972; Sacerdoti, 1977]). In behavioral based approaches, it is common to allocate (or assume the existence of) sensory processes that continually monitor all relevant sensory attributes (e.g., [Brooks, 1986; Kaelbling, 1987]). While this approach may be appropriate for relatively simple tasks, where the number of relevant attributes is small and well known, it fails to scale to more complex tasks.

In particular, as the number, complexity, and diversity of tasks performed by a robot increase, the number of potentially relevant features needed for decision making explodes, and it quickly becomes impossible to monitor continually every attribute that is potentially relevant. The problem is exacerbated when the task to be performed is not well understood ahead of time, since in this case the set of potentially relevant attributes is *potentially* unbounded.

Under these circumstances an active approach to perception is more appropriate. The principal idea of active perception is to control the allocation of sensory processing resources in order to analyze selectively only those aspects of the environment that are *actually* relevant to the immediate decision at hand. Following this approach, generality is attained by using a powerful set of sensory processing resources that can be flexibly applied to different parts of the environment and combined to define complex perceptual functions. Efficiency is attained by selectively applying these resources only as needed to satisfy the immediate information requirements of the internal decision system.

With respect to task performance, the idea is that as the robot progresses through different stages of a task or proceeds from one task to another, the changing information needs of the robot are tracked by actively controlling the computations performed by the sensory system. The degree to which the set of relevant features changes over time depends largely on the range of tasks being performed, their complexity, and the capabilities of the internal decision system to maintain internal state (or memories). For instance, a robot that performs two very similar tasks may find that, to a large degree, both tasks share the same set of relevant features, whereas a robot that performs two dissimilar tasks may find that the relevant features for the two tasks are nearly disjoint. For complex tasks, changes in relevant information may accompany transitions between stages of the task. The amount of context (or memory) maintained by a robot also determines the dynamics of systems information requirements. A robot that maintains information about its previous states may only need to verify that its most recent action had the intended effect, whereas a memoryless robot may need to use its sensors to reestablish context at each point in time.

With respect to learning, active perception provides a flexible means for sampling and monitoring a tremendous range of potentially relevant features (needed during learning), while it also provides for the efficient generation of task-dependent internal representations once the relevant sensory information has been discovered.

The key assumption exploited by the active perception paradigm is that at any point in time the number of features actually relevant to an agent's immediate decision is relatively small, even though the set of features that are potentially relevant (or relevant at some other point in time) may be large. This assumption appears to become universally valid as the number, complexity, and diversity of tasks to be learned and performed by an agent increase.

2.2 Visual Routines

In 1984, Shimon Ullman [Ullman, 1984] proposed an abstract computational model of human intermediate level vision. The model was developed to explain how the perception of spatial properties and relationships that are complex from a computational standpoint nevertheless often appear deceivingly immediate and effortless for humans. The distinguishing feature of Ullman's model is that complex spatial analysis is performed by a set of sequential processes called *visual routines*. With the visual routines model, Ullman made several important contributions to the study of visual perception:

1. He argued for active, top-down control and the selective application of visual processing resources, a significant departure from many of his contemporaries, as the prevailing dogma emphasized bottom-up scene reconstruction.

2. He recognized that many abstract spatial properties and relations have a certain amount of *essential sequentiality* and are often best described (and perceived) using sequential algorithms.

3. He argued that if properly chosen, a fixed set of basic visual operations could be assembled to extract an unbounded variety of shape properties and spatial relations, and that sharing these operations could yield visual processing systems that were both efficient and tremendously versatile.

4. He proposed a plausible set of basic operations and demonstrated their utility on a number of difficult spatial reasoning tasks.

2.2.1 Model Overview

In the visual routines model, the computation of spatial relations is divided into two stages. The first stage involves the bottom-up creation of low level base representations. The second stage involve the application of visual routines to the base representations to extract useful spatial properties.

The base representations are derived strictly bottom up. They are assumed to be spatially uniform, viewer centered, and unarticulated. Examples of plausible base representations include the primal sketch and the $2\frac{1}{2}$-D sketch as described by Marr [Marr, 1976; Marr and Nishihara, 1978]. Local information, such as depth, color, edge orientation, curvature, motion, and texture, is represented in the base representations.

The second stage of processing involves the application of visual routines to the base representations. The particular visual routine applied in a given situation is task-dependent and determined by the information needs of the higher level (decision making) components of the system. Indeed, Ullman suggests that visual processing be viewed as a query-answering process. Following this view, higher level decision making components posit queries to the visual system. These queries get translated into visual routines which are applied to the base representations. The results of this processing are then made available to the higher level components via a central representation, a functional analog to the internal representations of classical planning.

The processing stages of the visual routine model are depicted in Figure 2.1.

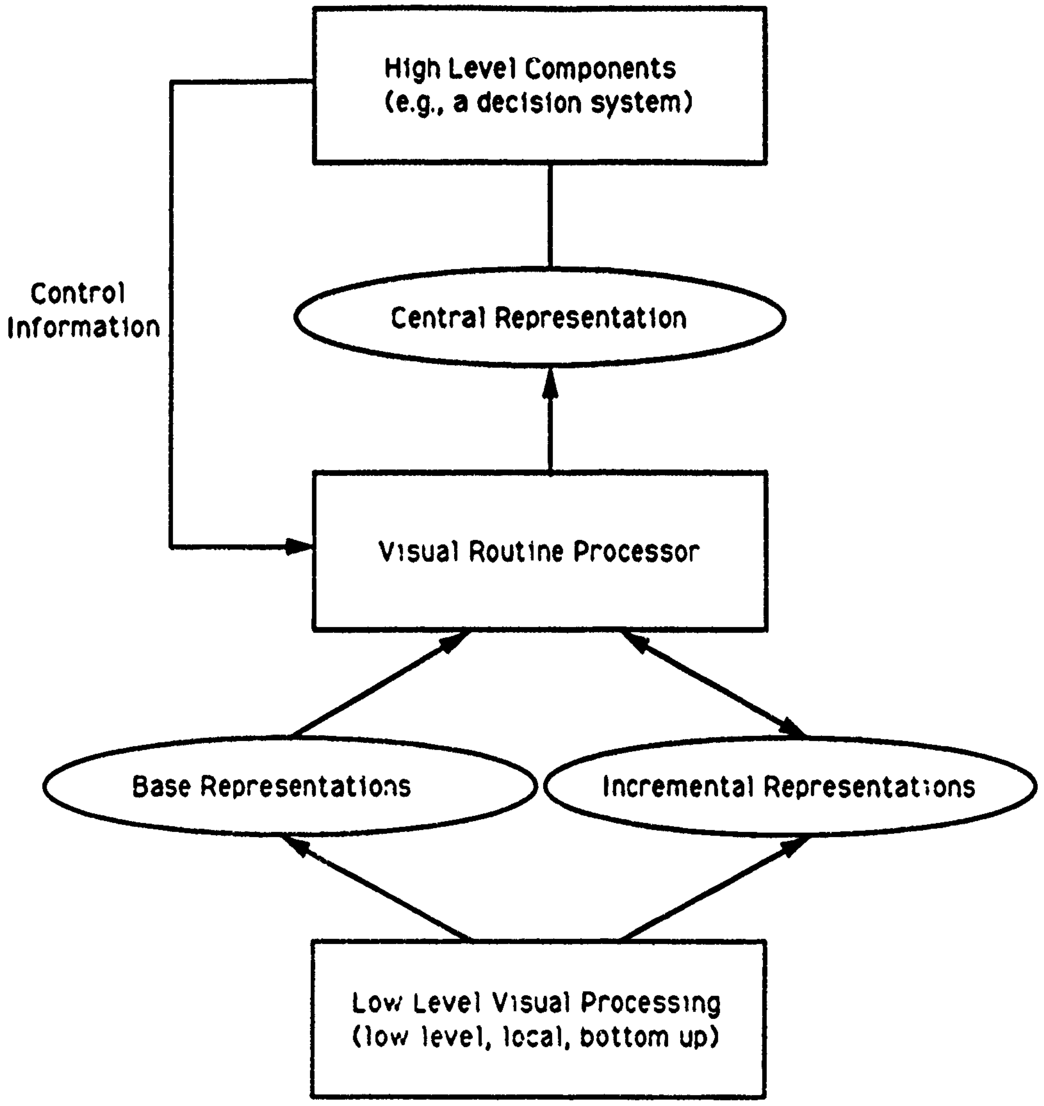

High Level Components
(e.g., a decision system)
Control
Information
Central Representation
Visual Routine Processor
Base Representations
Incremental Representations
Low Level Visual Processing
(low level, local, bottom up)

Figure 2.1: The processing stages of the visual routines model. The base representation is generated by low-level, local visual processes; visual routines are applied to the base representation, as determined by top-down feedback from higher level components. The results of visual routine processing are placed in a central representation for use by higher level decision-making components. Intermediate results may also be stored in an incremental representation and used for later processing.

2.2.2 Operations

Visual routines are sequential programs composed from a fixed set of basic operations. These operations are assumed to be implemented in hardware and shared between routines. Ullman has suggested a set of possible operations, based on their potential usefulness, and demonstrated their utility on a number of examples. These operations include: *shift of processing focus*, *indexing to an odd-man-out location*, *bounded spreading of activation*, *boundary tracing*, and *marking*. Following is a brief description of each of these operations (as taken almost verbatim from [Ullman, 1984] p. 155).

> *Shift of the processing focus.* This is a family of operations that allow the application of the same basic operation to different locations across the base representation.
>
> *Indexing.* This is a shift operation towards special odd-man-out locations. A location can be indexed if it is sufficiently different from its surroundings in an indexable property. Indexable properties include contrast, orientation, color motion, and perhaps also size, binocular disparity, curvature, and the existence of terminators, corners, and intersections.
>
> *Bounded activation.* This operation consists of the spreading of activation over a surface in the base representation, emanating from a given location or contour and stopping at discontinuity boundaries. This is not a simple operation, since it must cope with difficult problems of noise, spurious internal contours, and fragmented boundaries.
>
> *Boundary tracing.* This operation consists of either the tracing of a single contour or the simultaneous activation of a number of contours. The operation must be able to cope with the difficulties raised by the tracing of incomplete boundaries, tracing across intersections and branching points, and tracing contours defined at different resolution scales.
>
> *Marking.* The operation of marking a location means that this location is remembered, and processing can return to it whenever necessary. Such operations are useful in the integration of information in the processing of different parts of a complete scene.

2.2.3 Examples

The utility of visual routines and the plausibility of the above operations can be demonstrated by considering a number of visual tasks. Three tasks examined by Ullman are shown in Figures 2.2-2.4. Figure 2.2 shows several examples of the

inside-outside task. In this case, the objective is to determine visually if the X is contained by a closed curve. A visual routine for computing the inside-outside relation, based on "region coloring," proceeds as follows:

1. **Move the processing focus to the X location.**

2. **Begin a bounded activation at X until no further spreading occurs.**

3. **If the activation fails to reach the edge of the visual field, then report that X is contained by a closed curve.**

Another spatial reasoning task that is amenable to visual routines but difficult for classical pattern recognition techniques, is to determine whether or not two X's fall on the same curve (see Figure 2.3). A visual routine for establishing this property follows:

1. Move the processing focus to an unmarked X and mark the location.

2. If the X does not lie on a curve, then go to Step 1.

3. Trace along the contour until another X is encountered or the curve has been completely scanned.

4. If a second X is encountered along the contour in Step 3, then terminate and report success; otherwise, if unmarked X's still exist, go to Step 1; else terminate and report failure.

Interestingly, when human subjects perform this task, they report their perceptions as immediate and effortless when in fact reaction time experiments show that the time needed to perform such tasks monotonically increase (nearly linearly) with the distance traced along the contour [Ullman, 1984]. This is consistent with the tracing routine given above and suggests that, although unconscious of it, people probably use a sequential algorithm that includes a contour tracing operation.

A third example of a visual task is to identify a subfigure in a scene despite the presence of confounding figures in close proximity to its contours (see Figure 2.4). This task is representative of recognition problems found in natural scenes where cluttered backgrounds are common. The task can be accomplished by using a bounded activation process to segment the shape, followed by a shape analysis procedure.

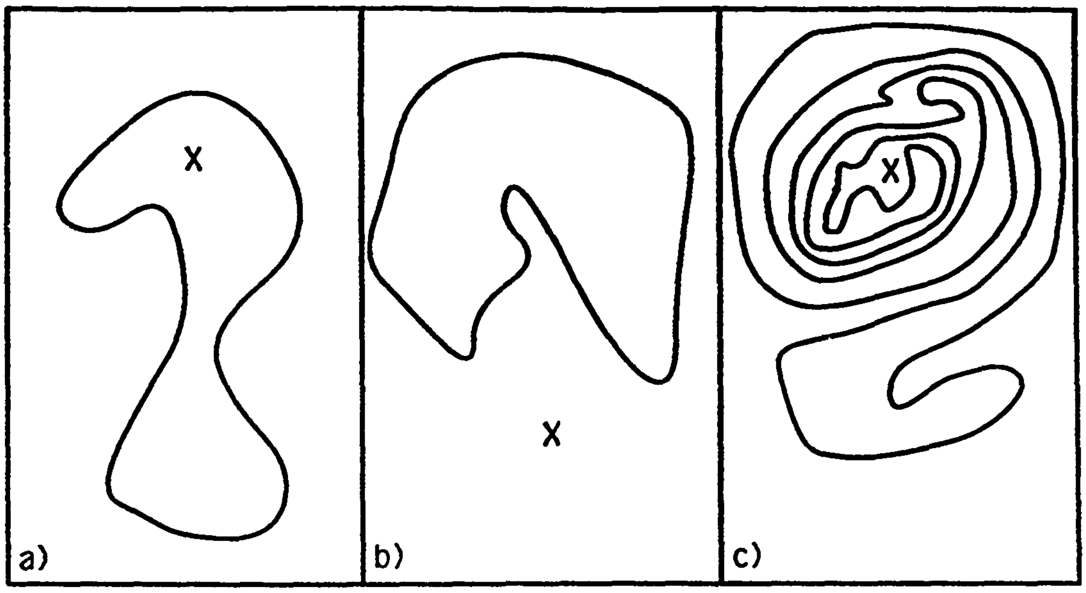


Figure 2.2: Examples of the inside-outside task (after [Ullman, 1984] p. 100). Two simple instances of the inside-outside task are shown in (a) and (b). A more difficult instance is shown in (c).

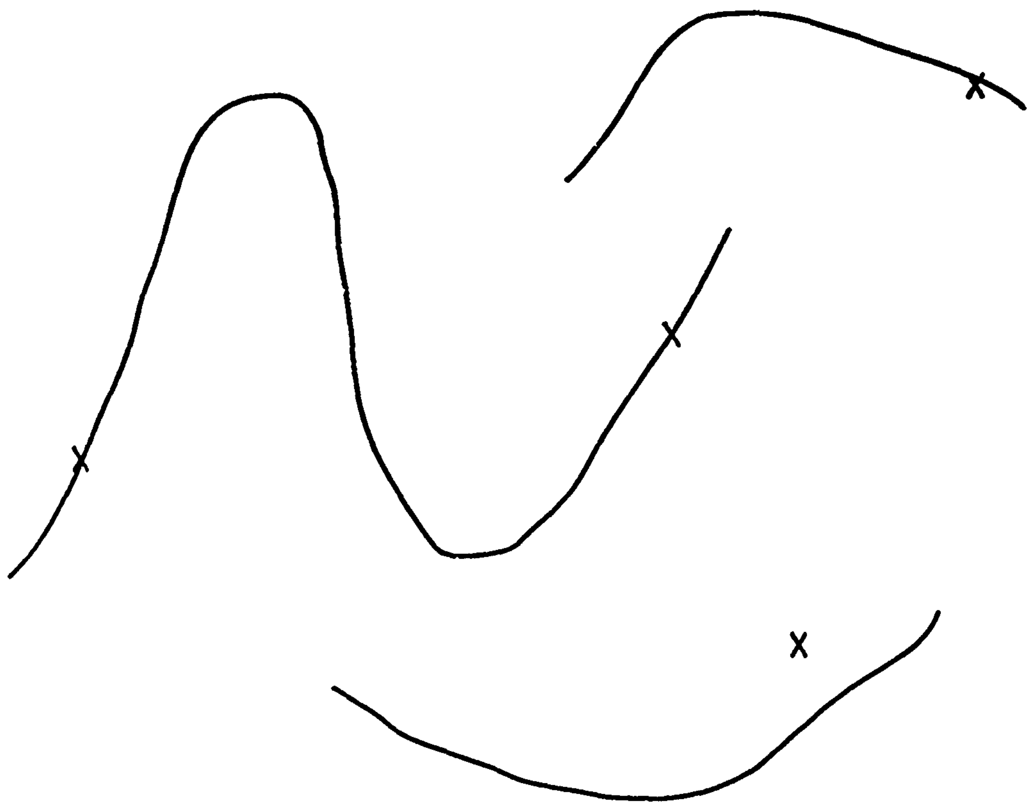


Figure 2.3: The objective of this task is the determine whether or not two X's lie on the same curve. Although human subjects report their perception of this property as immediate and effortless, reaction time experiments implicate a contour tracing operation in their computation.

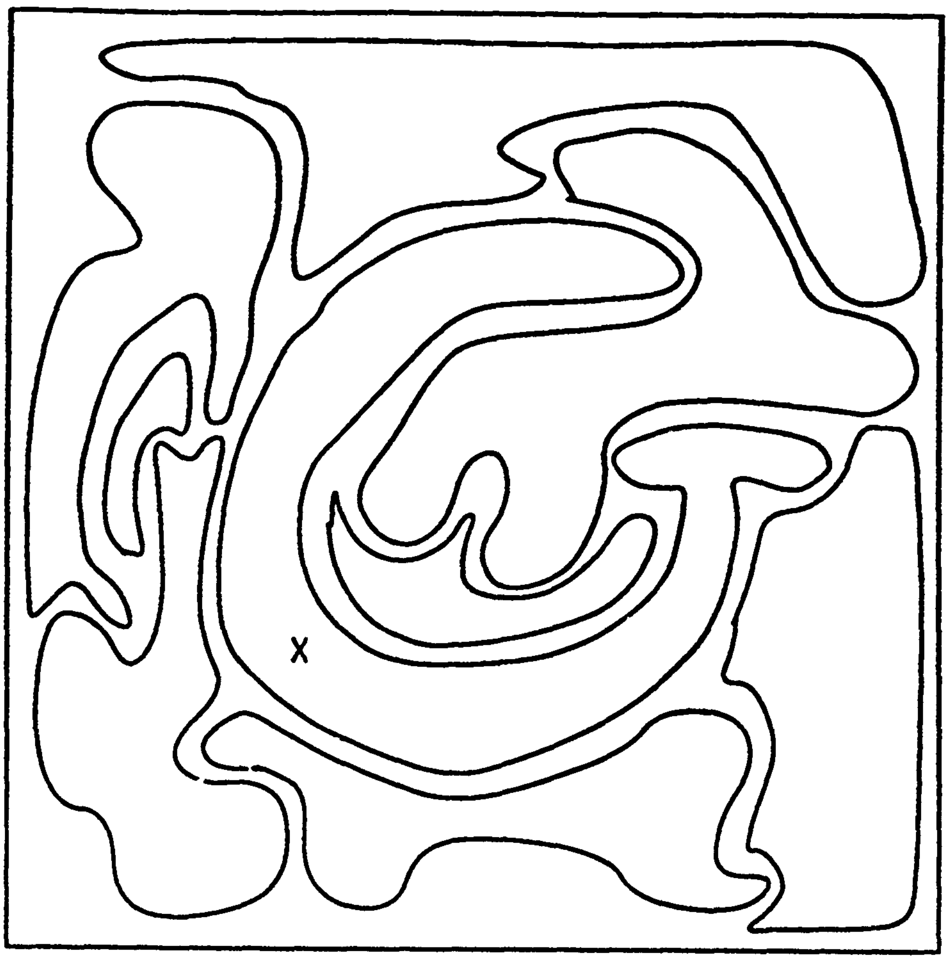


Figure 2.4: The objective of this task is to identify the subfigure containing the X despite the presence of confounding figures in close proximity to its contours. The key to this task is to segment the subfigure from the cluttered background. This can be accomplished by shifting the processing focus to the X and initiating a bounded spreading of activation (after [Ullman, 1984] p. 136).

2.2.4 Indexing

There are two essential forms of selectivity in the visual routines model: routine selection and indexing. Routine selection is the process of constructing and selecting the visual routines to be applied at a given point in time. Other than to associate it with higher level components of the system, algorithms for routine selection are not specified by the model. Indexing is the process of determining where in the base representation to apply a visual routine.

Examination of the above visual routines shows that processing does not occur in parallel over the entire base representation, but tends to be localized in specific, relevant regions. For instance, the algorithm for finding two X's on a common contour begins by focusing processing at locations occupied by an X. These locations in the base representation act as "anchor points" for the routine and establish context-dependent referents used by the visual routine.

In the visual routines model, indexing is accomplished in three stages. In the first stage, a set of indexical properties are computed in parallel over the entire base representation. These indexical properties are assumed to be locally computable features such as motion, color, orientation, and curvature. In the second stage, an odd-man-out operation is performed to detect locations that are significantly different from their surroundings. This salience operation is mediated by selecting specific indexical properties to be emphasized, or satisfied by the chosen anchor point. In the last stage, the processing focus is shifted to the most salient location.

A cornerstone of the active vision paradigm in general, and the visual routines model in particular, is the idea of limiting the application of visual processing resources to only those spatial locations that are functionally relevant. From a computational standpoint, the viability of this idea hinges on being able to quickly identify functionally relevant locations in the base representation. An important assumption made by the visual routines model is that in most cases locally computable features can be used to quickly identify these functionally relevant regions. Whether or not this assumption stands up in practice remains to be seen. However, it appears that in many cases local properties can indeed be used to differentiate relevant locations from the background. From a psychological standpoint, considerable evidence exists to suggest that the human visual system employs some indexing mechanisms based on locally computable features [Treismann and Gelade, 1980; Julesz, 1981]. Recent work by Swain, Ballard, and Wixson has also demonstrated the utility of locally computable properties based on color for indexing in an active computer vision system [Swain, 1990; Wixson, 1990; Wixson and Ballard, 1991].

2.2.5 Summary of the Visual Routines Model

The human visual system has an uncanny ability to perform, with surprising ease, spatial analysis tasks that from a computational standpoint are quite sophisticated. Few computational models are capable of explaining such sophisticated processes. Ullman's visual routine model represents a significant advance in this regard. It has also contributed to the recent shift in computational vision away from scene reconstruction and towards active, task-oriented perception.

The key features of the Visual Routines Model follow ([Ullman, 1984] p. 108):

1. Spatial properties and relationships are established by the application of visual routines to a set of early visual representations.

2. Visual routines are assembled from a fixed set of elemental operations.

3. New routines are assembled to meet newly specified processing goals.

4. Different routines share elemental operations.

5. A routine can be applied to different spatial locations. The processes that perform the same routine at different locations are not independent.

6. Mechanisms are provided for sequencing elemental operations and for selecting locations to apply routines.

2.3 Deictic Representations

One area that Ullman's visual routines model does not address in detail is the interface between the visual routine processor and higher level control components of the system. In particular, Ullman's model does not answer these questions: What information should be encoded in the internal representations of the higher level components? or How do the higher level components control the construction and application of visual routines?

Agre and Chapman have proposed one answer to the question of the content of the central representation in their theory of *deictic representations* [Agre and Chapman, 1987; Agre, 1988; Chapman, 1990b]. The key observation exploited in this theory is that for any particular task (or activity) there is usually, at any given point in time, a relatively small number of objects that are immediately relevant to the agent's behavior. Instead of attempting to represent each and every object in the domain objectively (as is common in traditional representational schemes), deictic representations aim to monitor and represent actively *only* those few key objects that are immediately relevant to the ongoing activity. In a deictic representation, if some aspect of the world is not significant to the agent's immediate activity it is ignored. This task-oriented approach to representation has the advantages that 1) it significantly reduces the computational burden placed on the sensory system and 2) it affords a kind of "passive abstraction" [Agre, 1988] that greatly facilitates decision-making by collapsing the infinite complexity of the real world onto much smaller task-specific internal representations.

2.3.1 Entities and Aspects

In a deictic representation, it is assumed that a task (or activity) can be described abstractly in terms of continuous interactions with, and manipulations of, abstract functional objects called *indexical functional entities.* That is, it is assumed that knowledge of relevant features and relationships of these few key entities is sufficient to determine the behavior of an agent. These abstract properties and relationships are called *indexical functional aspects* and they comprise the agent's internal representation.

A simplistic (but intuitive) way to think about entities is to view them as functional roles that must be instantiated by objects in the world in the course of actually performing a particular task. That is, when an agent actually engages in an activity, entities get bound to specific objects in the world, and the actual behavior of the agent (in a particular instantiation of an activity) is determined by the actual properties and relationships (aspects) of these bound objects. At any given point in time, an entity is associated with at most one object in the real world. However, over the course of time, in various instantiations of an activity or in different stages of an activity, an entity may bind many different objects. The point is that entities correspond to functional roles in an activity, which may be played by a wide range of objects, depending upon the particular circumstances in the external world. In this discussion, we focus on indexical functional entities that represent physical objects in the external world. These simple entities are analogous to natural kinds (or simple nouns) used in linguistics. However, just as there are nouns to describe complex, abstract concepts, so too can complex indexical functional entities be defined to represent them.

Aspects describe important properties of entities. These may include relatively simple features, such as color, texture, orientation, and position in space, or arbitrarily complex properties and relationships such as shape, relative size, inside-outside relationships, overlap, alignment, distance, or just about any condition imaginable. In addition to determining the overall state of the agent with respect to a task, aspects are also useful for instantiating specific motor commands. For instance, the position of an entity relative to the agent's body may be used to establish the reference frame used during reaching [Ballard, 1989b].

2.3.2 Implementing Deictic Representations

In general, in a deictic representation, there are many ways to establish bindings between entities and aspects in the agent's internal representation and specific objects and properties in the external physical world. For instance, the binding between an entity such as *the-cup-from-which-I-am-drinking* and a physical object (say the actual cup I am drinking from) can be established through visual cues (e.g., visual fixation), through haptic cues (e.g., grasping it with one's hand), or even through memory (e.g., by remembering the values of aspects/properties that characterize it).

It is also important to point out that, it is not necessary for the agent to continually maintain every object-entity binding associated with a specific activity. Bindings may be dynamic and may depend upon the status of the agent's ongoing activity and upon conditions in the world. In general, what is required between an object and an entity is that the agent maintain a causal relationship between the two so that when the time comes to establish/exploit the binding, it can be made. Such causal relationships can be maintained through behavioral conventions, habits, policies, laws, juxtapositions, *etc.* See Chapter 7 of [Agre, ming] for a complete discussion.

For the purposes of this dissertation, we restrict ourselves to simple cases where entities are always simple (i.e., bound to physical objects in the external world) and vision alone is used to establish object-entity bindings. In the visual systems described in [Agre, 1988; Chapman, 1990b], an object is bound to entity using a mechanism known as a *marker*. Markers are best thought of as pointers implemented by the visual system and are quite similar to the "processing foci" describe in Ullman's model. A marker can be bound to only one object at a time and it is assumed that the sensory-motor system maintains the marker's binding at all times. Changing a marker's binding is accomplished by executing explicit actions specifically targeted for that marker. These actions index target objects in the world according to specific indexical properties that distinguish them from other objects, as in Ullman's model.

It is important for an agent to establish object-entity bindings as quickly and efficiently as possible. To facilitate binding, each entity has associated with it a set of indexical properties/routines that are used to guide the search for candidate objects. Once a candidate object has been identified, additional aspects can be computed using more complex visual routines to determine whether or not it meets the functional constraints of the entity.

2.3.3 Control

As in Ullman's model, Agre and Chapman's theory assumes that higher level (decision) components control the selection of visual routines and, consequently, the generation of thc agent's internal representation. In the systems they implemented, the higher level decision components were hand-crafted combinational circuits, and other than to emphasize the need for continuous interaction between perception and control, no formal theory of high level control was provided.

2.3.4 Instantiations of the Theory

Agre and Chapman wrote a pair of programs that, among other things, demonstrated the utility of the visual routines model and of deictic representations in complete behaving systems[Agre and Chapman, 1987; Agre, 1988; Chapman, 1990b]. The programs play two different video games. The first program, called Pengi, plays Pengo; and the second program, called Sonja, plays Amazon. I'll focus on Pengi.

Pengo is an interactive video game in which a penguin moves around in a maze of ice blocks that is also occupied by bees. The player (Pengi) controls the penguin's movements with a joystick while the bees fly around semi-randomly. The penguin can be stung and killed by bees that get too close. Also, the penguin and bees can modify the maze by kicking ice blocks to make them slide. If a block slides into a bee or the penguin it dies. Figure 2.5 shows a snapshot of a Pengo game in progress. In the figure, the penguin can be found in the middle left part of the screen.

In Pengi, entities and aspects are used to identify ice cubes and bees (and other things like corridors and openings) that are relevant to the player's behavior. At various times during play, Pengi's internal representation monitors different entities and aspects, depending upon the particular activity engaging Pengi. Some of the entities that Pengi keeps track of at various times during the game are:

the-ice-cube-I-am-kicking,
the-bee-I-am-chasing,
the-bee-on-the-other-side-of-this-ice-cube-next-to-me,
the-ice-cube-that-the-ice-cube-I-just-kicked-will-collide-with.

Space does not permit a detailed discussion of the visual routines used to identify these entities or compute their aspects. However, elegant discussions of the theory of deictic representations and its realization in Pengi and Sonja can be found in incarnations of Agre's and Chapman's doctoral dissertations, [Agre, 1988; Agre, ming] and [Chapman, 1990b; Chapman and Kaelbling, 1991], respectively.

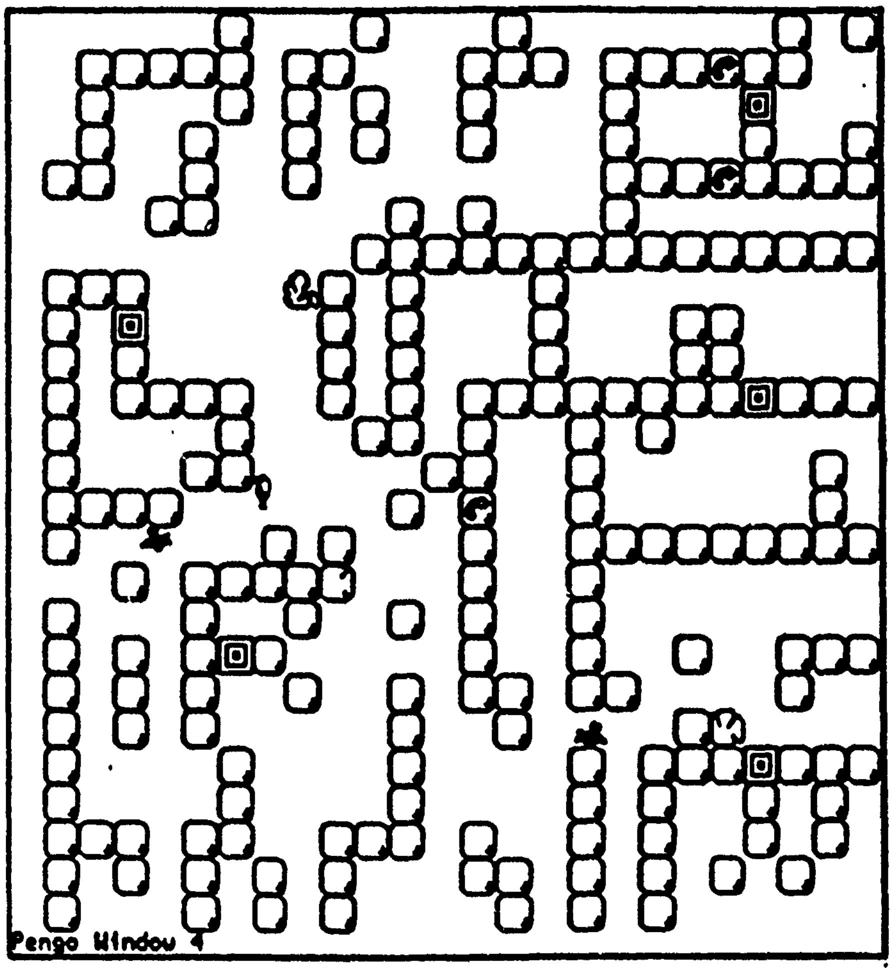


Figure 2.5: A Pengo game in progress (from [Agre, 1988] p. 196).

3 Background: Reinforcement Learning

This chapter reviews some of the principles of reinforcement learning. The main objective of the chapter is to review Watkins' Q-learning algorithm; however, we would also like to establish some intuition into the workings Q-learning and relate it to the mathematical foundations on which it is based. Therefore, we begin by reviewing the Theory of Markov Decision Processes and the fundamentals of Dynamic Programming. We then show how Q-learning can be derived as a kind of incremental, Monte Carlo version of the policy iteration algorithm of dynamic programming. More thorough treatments of the Theory of Markov Decision Processes and Dynamic Programming can be found in numerous good books (e.g., [Bellman, 1957; Ross, 1983; Bertsekas, 1987]) and an excellent account of Q-learning can be found in Watkins' PhD dissertation [Watkins, 1989].

3.1 Markov Decision Processes

We are interested in agents that learn to perform tasks that require interactions with the world over an extended period of time. These tasks require the agent to use sensors to differentiate states of the world and to decide, depending upon the situation, which action to take. Sometimes actions will achieve a desired end (or a goal) but most of the time actions will be used to preserve some desired property of the world or to set up opportunities to achieve goals in the future.

Markov decision processes provide a useful mathematical framework to describe these sequential decision tasks. In a Markov decision process (also called a *controllable Markov chain*) a control task is formally modeled by the tuple $(\mathbf{S}, \mathbf{A}, T, R)$. In this model, $\mathbf{S}$ is the set of possible states the world can occupy, $\mathbf{A}$ is the set of possible actions the agent (or controller) can take, T is a transition function that determines the effects of actions, and R is a reward/cost function that associates payoffs and penalties with actions and states. In this dissertation we will assume that the set of possible world states, $\mathbf{S}$, and the set of available actions, $\mathbf{A}$, are discrete and finite.

In a Markov decision process, time advances in unit duration quanta ($t = 0, 1, 2, 3, 4, \ldots$), where at each point in time the system occupies exactly one state.[1] At each point in time, the agent selects and executes one action, which causes the world to make a transition to a new state and results in the receipt of a reward (or penalty). At any given time t,

> X_t is the random variable denoting the state of the system,
> x_t is the actual state of the system,
> R_t is the random variable denoting the reward received, (i.e., as a result of executing an action in state X_t),
> r_t is the actual reward received,
> a_t is the action executed by the controller.

The effects of an action depend only upon the state in which it was performed. This dependence is modeled by the transition function T, such that $T(x_t, a_t) = X_{t+1}$. For deterministic models, the mapping from state-action pairs to next states is fixed and the transition function is a true function. However, in general transitions are allowed to be probabilistic. In this case, the result returned by T is a sample drawn from a probability distribution over **S**. In a Markov decision process the probability distributions that govern the transition function depend *only* upon the action and the state in which it was performed, and the transition function is fully specified by a set of transition probabilities $P_{xy}(a)$, where

$$P_{xy}(a) = Pr(T(x, a) = y). \tag{3.1}$$

Similarly, the reward function R determines the reward received as a result of executing an action. It, too, is a probabilistic function of the current state-action pair, thus, $R_t = R(x_t, a_t)$. While a set of probability distributions governing the reward function could be defined for each state action pair, analogous to the transition function, we are usually only concerned with the expected reward obtained as a result of executing an action. Thus, we will assume instead that the Markov Decision Process defines an expected reward with each state action pair, which we write as

$$\rho(x, a) = \mathbf{E}[R(x, a)]. \tag{3.2}$$

This completes the formal definition of a Markov Decision Process.

3.1.1 The Markov Property

A key property of the Markov Decision Process model is that transitions and rewards depend only upon the current state and the current action. That is *not*

[1] Actually, continuous time and continuous state Markov decision processes can be defined. However, our discussion will restrict itself to discrete time and discrete state processes.

to say that knowledge of the current state and current action is sufficient to predict the reward and next state — transitions and rewards may be non-deterministic — but that any information above and beyond knowledge of the current state and current action is useless for making predictions. This is called the *Markov property*. It is a very strong condition because it says that knowledge of the current state-action pair captures the essence of the task and that any additional knowledge of the world whatsoever has no impact on one's ability to predict the outcome of executing the state-action pair.

It is important to point out that the Markov property is not an intrinsic property of any physical process; rather it is a property of a *mathematical model* of a physical process. Virtually any physical process can be modeled as a Markov process if the state-space and control actions are chosen appropriately.

The Markov property is crucial when considering the effects of active perception on the decision problems faced by an agent's embedded control system. In particular, if the state space is defined by the possible values of the agent's sensory inputs and if the sensory system does not provide adequate information about the state of the world, then the decision problem facing the agent's embedded controller may not satisfy the Markov property. As we will see in Chapter 5 this can be catastrophic since 1) it may lead to decision problems that have no fixed optimal policy and 2) it may cause reinforcement learning algorithms to oscillate from one policy to another, never converging on an optimal strategy.

3.1.2 The Objective of Control

Policies

Markov decision processes are also called Controllable Markov Processes because one essential component of these models is a controller that, by choosing actions, can influence (and in some cases directly control) the state space trajectory of the world through time. One way to specify the behavior of the controller is with a *policy*. A policy is a rule that determines the action to execute given the current state. Formally, a policy f is a function from states to actions ($f : \mathbf{S} \to \mathbf{A}$), where $f(x)$ denotes the action to be executed in state x.

A policy is *stationary* if the mapping from states to actions is fixed. If the function is probabilistic then the policy is said to be *stochastic*, otherwise it is *deterministic*. Given a Markov decision process, the objective is to find (either by direct computation or by learning) a deterministic, stationary policy that is in some sense optimal.

Let us denote the probability that the process makes a transition from state x to state y, given that the agent is following policy f, as

$$P_{xy}(f). \tag{3.3}$$

Similarly, let us use

$$T(x,f), \quad R(x,f), \quad \text{and} \quad \rho(x,f) \tag{3.4}$$

to denote the random variables for the next state, the reward and the expected reward, respectively, when following policy f in state x.

Returns

In the Theory of Markov decision processes, optimality is defined in terms of cumulative reward received over time. That is, the goal of the agent is to implement a control policy that on average maximizes some measure of the total reward to be received in the future.

There are several possible measures of cumulative reward. One of the most common is a measure based on a discounted sum of the total reward received over time. This sum is called the *return* and for time t is defined as

$$\mathbf{r}_t = \sum_{n=0}^{\infty} \gamma^n r_{t+n} \tag{3.5}$$

where γ, called the discount factor, is a constant between 0 and 1. In this measure, the discount factor is used to weight the importance of a reward as a function of its distance into the future. Setting $\gamma = 0$ leads to shortsighted policies that are interested only in the most immediate reward. Setting the discount factor near one leads to returns that weigh rewards some time into the future and results in optimal policies that may forgo immediate rewards in order to set up larger rewards down the road. For any $\gamma < 1$ the effects of rewards far into the future eventually become negligible and the return is finite. Because a process may be stochastic, the objective is to find a policy that maximizes the *expected return.*

Optimality Criterion

For a fixed policy f, define $V_f(x)$ to be the expected return, given that the process begins in state x and follows policy f thereafter.

To define V_f more precisely we must introduce more notation. Let $X(x,f,n)$ be a random variable denoting the state that results from starting in state x and following policy f for n steps. Note that $X(x,f,0) = x$ and $X(x,f,1) = T(x,f)$. Similarly, let $R(x,f,n)$ be the random variable denoting the reward received at time $t+n$ after starting in state x and following policy f for $n+1$ steps. So, for example, $R(x,f,0) = R(x,f)$. Also, we will sometimes use $X(x,a,1)$ as alternative notation for $T(x,a)$. Given these definitions, V_f is defined as

$$V_f(x) = \mathrm{E}[R(x,f,0) + \gamma R(x,f,1) + \gamma^2 R(x,f,2) + \ldots + \gamma^n R(x,f,n) + \ldots]. \tag{3.6}$$

V_f is called the ***value function*** for policy f and it can be used to define an optimality criterion. Namely, given a description of a Markov decision process, the objective is to find a policy whose value function is uniformly maximal for every state. That is, find a policy f^* such that

$$V_{f^*}(x) = \max_f(V_f(x)) \;\; \forall_{x \in \mathbf{S}}. \tag{3.7}$$

Intuitively, an optimal policy is any control strategy that always maximizes the expected return, no matter what state the process finds itself in and no matter what the history of the process may be.

3.1.3 Example

To illustrate how sequential tasks can be formulated as Markov decision processes, consider the task shown in Figure 3.1. In this task, we want the robot to navigate from an initial state S, through the maze, to the goal state G. The robot can take unit length steps in each of the four principal compass directions and cannot pass through barriers. For simplicity we will assume that actions are deterministic.

A formulation of this task as a Markov decision process is shown in Figure 3.2. In this formulation the robot receives a positive reward only when it executes the action that causes it to enter the goal state. In general the reward function must be carefully chosen if the optimal policy associated with the decision process is to match the desired behavior. One particularly convenient approach is to identify a set of desirable goal states and to reward the agent whenever it encounters one of these states. Following this approach, the optimal policy usually corresponds to taking actions that achieve the goal in the least number of steps. Of course for a given task, or behavioral interaction, there are an infinite number of formulations whose optimal policies will yield the desired behavior.

The optimal policy and value functions for the formulation given in Figure 3.2. is shown in Figure 3.3. The optimal policy is indicated with an arrow in each square. The values for V_{f^*} are shown in the lower right corner of each tile. These values are for $\gamma = 0.9$. Notice that the policy is defined over the entire state space and that following the policy from any location traces out a shortest path to the goal state. Also notice that the value function monotonically increases as the robot approaches the goal state. Indeed, for this formulation of the task, the optimal trajectory follows the gradient in the value function.

3.2 Dynamic Programming

At this point Markov decision processes have been defined and shown to be useful for formalizing sequential control tasks. However, a crucial question has not been

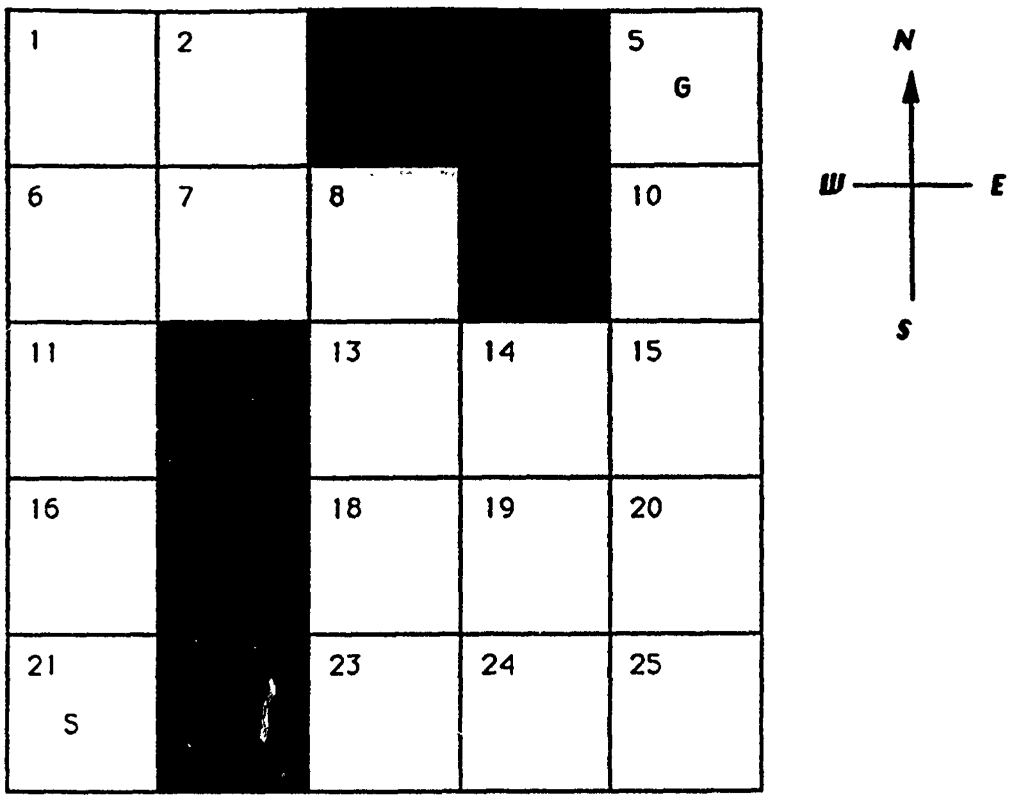


Figure 3.1: The navigation task, a simple example of a sequential control problem. The objective of the robot is to navigate from an initial location S to a goal state G as quickly as possible.

$$R(x,a) = \begin{cases} 1 \text{ if } x = 10 \text{ and } a = N \\ 0 \text{ otherwise} \end{cases}$$

A = { N, S, W, E}
S = { 1, 2, 5, 6, 7, 8, 10, 11, 13, 14,
15, 16, 18, 19,. 20, 21, 23, 24, 25}

T(x,a) :

A \ S	1	2	5	6	7	8	10	11	13
N	1	2	5	1	2	8	5	6	8
S	6	7	5	11	7	13	15	16	18
E	2	2	5	7	8	14	16	11	14
W	1	1	5	6	6	8	10	11	13

A \ S	14	15	16	18	19	20	21	23	24	25
N	14	10	11	13	14	15	16	18	19	20
S	19	20	21	23	24	25	21	23	23	25
E	15	15	16	19	20	20	21	24	25	25
W	13	14	16	18	18	19	21	23	23	24

Figure 3.2: A formalization of the navigation task as a Markov decision process. Shown is the state space S, the set of possible actions A, the reward function R, and the transition function T. In this model, there are 19 states, 4 actions, and the agent receives a reward when it first enters the goal state.

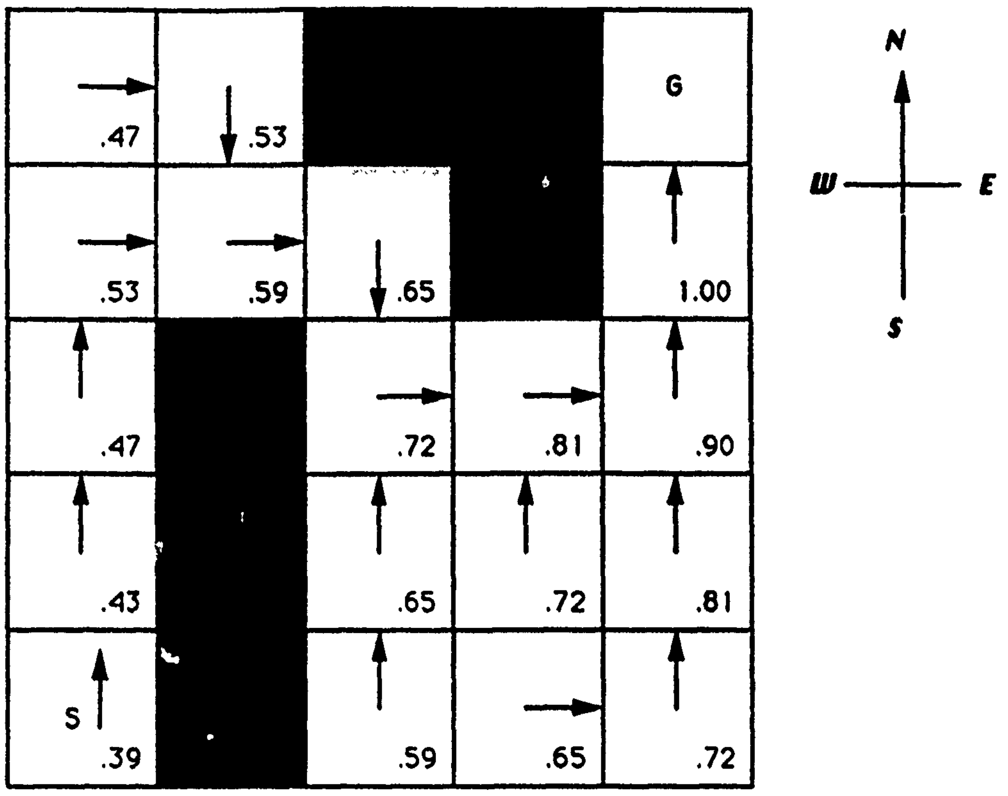


Figure 3.3: The optimal policy and optimal value function for the navigation task. The direction of the arrow indicates an optimal action to take when occupying a given tile. Values for the optimal value function are shown in the lower right corner of each tile. Notice that the value function is inversely proportional to the distance to the goal and that the optimal policy follows the gradient in the value function.

addressed: How do we find an optimal policy? If all of the parameters of the Markov decision process are known precisely, then Dynamic Programming can be used to compute an optimal policy. This section reviews essential ideas and techniques of dynamic programming.

3.2.1 The Brute Force Method

One way to obtain the value function for a given policy is to estimate it by Monte Carlo simulation. That is, $V_f(x)$ can be estimated accurately by repeatedly starting the process off in state x, letting it follow policy f for some (possibly long) period of time, and accumulating a return for each trace. Since each trace is independent, the law of large numbers guarantees that an average of the trace returns will eventually converge to the expected value ($V_f(x)$).

A better way to compute the value function is to compute it directly by noticing that it can be defined recursively. In particular, the definition of $V_f(x)$

$$V_f(x) = \mathbf{E}[R(x,f,0) + \gamma R(x,f,1) + \gamma^2 R(x,f,2) + \ldots + \gamma^n R(x,f,n) + \ldots] \tag{3.8}$$

can be written recursively as

$$V_f(x) = \mathbf{E}[R(x,f,0)] + \gamma \mathbf{E}[V_f(X(x,f,1))] \tag{3.9}$$

or equivalently as

$$V_f(x) = \rho(x,f) + \gamma \mathbf{E}[V_f(X(x,f,1))]. \tag{3.10}$$

For finite state processes, the expectation on the right hand side can be written as a sum over a finite number of states, leading to the expression

$$V_f(x) = \rho(x,f) + \gamma \sum_{y \in \mathbf{S}} P_{x,y}(f) V_f(y). \tag{3.11}$$

Equation 3.11 defines a family of linear equations which can be solved for V_f when combined with the constraint expressions

$$\sum_{y \in \mathbf{S}} P_{xy}(f) = 1 \forall_x \tag{3.12}$$

and the values for $\rho(x,a)$ and $P_{xy}(a)$ provided by the model.

Thus, given a finite state, finite action Markov decision process and any fixed policy, the value function for that policy can be computed directly; although solving the linear equations may be time consuming.

Given this means for computing a policy's value function, one way to find an optimal policy is to enumerate all possible policies, compute the value function for each, and choose one that is uniformly maximal. Unfortunately, this exhaustive approach is computationally intractable even for small problems since the number of possible policies grows exponentially in the state space size and the number of possible actions.

3.2.2 Policy Iteration

Dynamic programming offers a much cheaper (but generally still expensive) method for solving Markov decision problems. In dynamic programming, instead of comparing all possible policies in search of an optimal one, an arbitrary policy is initially selected and then incrementally improved until it is optimal.

Suppose we have two policies f and g, and we would like to know if g is uniformly better than f. One way to answer this question is to compute V_f and V_g as above and compare them directly. A less expensive method, which only requires the full computation of V_f, follows.

Suppose that in any initial state x, instead of always following policy f, we first follow policy g for *one step* and then switch back to f, which we follow thereafter. Consider the expected return that results from such a strategy. In particular, let $Q_f(x,b)$ denote the expected return, given that starting in state x the process executes action b and then follows policy f thereafter. Q_f is called the *action-value* function for policy f. The action-value function can be expressed in terms of V_f as follows:

$$Q_f(x,b) = \rho(x,b) + \gamma \sum_{y \in bfS} P_{xy}(b)V_f(y). \tag{3.13}$$

Given Equation 3.13, $Q_f(x,g(x))$ is much simpler to compute (given that we know V_f) than V_g.

Given the action-value function, we can determine if g is better than f. In particular, suppose that the expected return obtained by following g for one step and then switching back to f is uniformly as good as or better than the expected return obtained by following f only. That is, suppose that for all $x \in \mathbf{S}$,

$$Q_f(x,g(x)) \geq V_f(x). \tag{3.14}$$

Then by induction g is uniformly better than f. This can be seen by considering the expected return that results from following g for n steps and then switching back to f. Denote the resulting expected return as $Q_f(x,g,n)$. For $n = 1$,

$$Q_f(x,g,n) = Q_f(x,g(x)) = \rho(x,g(x)) + \gamma \sum_{y \in bfS} P_{xy}(g(x))V_f(y). \tag{3.15}$$

It is given that $Q_f(x,g(x)) \geq V_f(x)$, so for $n = 1$ we have

$$Q_f(x,g,1) \geq V_f(x) \quad \text{for all } x. \tag{3.16}$$

For the inductive step, assume $Q_f(x,g,n) \geq V_f(x)$ for all x. Now in general we can write

$$Q_f(x,g,n+1) = \rho(x,g(x)) + \gamma \sum_{y \in \mathbf{S}} P_{xy}(g(x))Q_f(y,g,n). \tag{3.17}$$

But since $Q_f(y,g,n) \geq V_f(y)$ for all y, it follows that

$$Q_f(x,g,n+1) \geq \rho(x,g(x)) + \gamma \sum_{y \in \mathbf{S}} P_{xy}(g(x))V_f(y). \tag{3.18}$$

The right hand side of this equation is just $Q_f(x,g,1)$, which by assumption is greater than or equal to V_f. So we have

$$Q_f(x,g,n+1) \geq V_f \quad \text{for all x and n.} \tag{3.19}$$

Of course, $V_g(x)$ is the limit as n goes to infinity of $Q_f(x,g,n)$, so $V_g(x) \geq V_f(x)$ for all x. This result is summarized by the *policy improvement theorem* [Bellman, 1957; Bertsekas, 1987; Watkins, 1989].

Theorem 1 (Policy Improvement Theorem)

Let f and g be policies, and let g be chosen so that

$$Q_f(x,g(x)) \geq V_f(x) \quad \textit{for all } x \in \mathbf{S}. \tag{3.20}$$

Then it follows that g is uniformly better than f. That is,

$$V_g(x) \geq V_f(x) \quad \textit{for all } x \in \mathbf{S} \tag{3.21}$$

The policy improvement theorem is a cornerstone of dynamic programming because it provides a relatively efficient means of finding better and better policies. Beginning with a policy f, a better policy f' can be found by 1) computing V_f, 2) calculating the action-value function $Q_f(x,a)$ for all state-action pairs $(x,a) \in \mathbf{S} \times \mathbf{A}$, and 3) defining f' at each state by choosing the action that maximizes the action-value function. That is, for all x,

$$f'(x) := a \quad \text{such that} \quad Q_f(x,a) = \max_{b \in \mathbf{A}} Q_f(x,b). \tag{3.22}$$

By the policy improvement theorem f' is guaranteed to be uniformly as good as or better than f.

This process can be repeated over and over again until, after a finite number of iterations, f no longer changes. At this point, the final policy f^* will satisfy the *optimality-equation*

$$f^*(x) = a \quad \text{such that} \quad Q_{f^*}(x,a) = \max_{b \in \mathbf{A}} Q_{f^*}(x,b). \tag{3.23}$$

Even though we know each iteration improves f, is it pos ·le that the terminal policy f^* is non-optimal? The *Optimality Theorem* tells us that indeed f^* is optimal. The following version of the Optimality Theorem is taken from [Watkins, 1989]; the proof of a similar version can be found in [Bellman, 1957].

```
f := an arbitrary initial policy
Repeat
    1. calculate the value function $V_f$,
    2. calculate the action-values $Q_f(x,a)$ for all $x,a$,
    3. update the policy for all $x$ by choosing $f(x) = a$ such that
        $Q_f(x,a) = \max_{b \in \mathbf{A}} Q_f(x,b)$
until there is no change in $f$ at step 3.
```

Figure 3.4: The Policy Iteration Algorithm.

Theorem 2 (Optimality Theorem)

Let a policy f^* have an associated value function V^* and an action-value function Q^*. If the policy f^* cannot be further improved by using the policy improvement theorem, that is if

$$V^*(x) = \max_{b \in \mathbf{A}} Q^*(x,b) \tag{3.24}$$

and

$$f^*(x) = a \quad \textit{such that} \quad Q^*(x,a) = V^*(x) \tag{3.25}$$

for all $x \in \mathbf{S}$, then V^* and Q^* are the unique, uniformly optimal value and action-value functions, respectively, and f^* is an optimal policy. The optimal policy f^* is unique unless there are states at which there are several actions with the maximal action-value, in which case any policy that recommends actions with the maximal action value according to Q^* is optimal.

A summary of the *policy iteration algorithm* is shown in Figure 3.4. This algorithm is guaranteed, via the policy improvement and optimality theorems, to converge on an optimal policy for any finite Markov decision problem.

3.2.3 Value Iteration

The policy iteration algorithm is an improvement over exhaustive search; however, it is still expensive. Note that the value function must be recalculated in each stage. Even though the value function for the new policy may be very similar to the old one, it cannot easily be derived from it, but instead must be computed by solving the set of linear equations given by Equations 3.11 and 3.12.

Value iteration is another computational technique for solving Markov decision problems that is often more efficient than policy iteration. In value iteration, instead of repeatedly computing the value functions for a series of ever improving non-optimal policies, the optimal policy is obtained by directly solving the optimality equation (Equation 3.23) for a series of finite-horizon tasks. As the horizon goes to infinity, the optimal policies of the finite horizon problems converge to the optimal policy for the original Markov decision problem.

In a finite horizon problem, the objective is find a policy that uniformly maximizes the expectation of a return that is truncated at the horizon boundary. For example, when the horizon equals one, $n = 1$, the goal is to find a policy that maximizes the expected reward received after one step. For $n = 2$, we want to maximize the cumulative reward received after two steps, and so on.

Let V^n denote the optimal value function for an n-step finite horizor problem. Since rewards are only received after executing an action, we have $V^0(x) = 0$ for all x. $V^1(x)$ can be determined by choosing the action that has the highest expected reward, namely

$$V^1(x) = \max_{a \in \mathbf{A}} \rho(x, a). \tag{3.26}$$

Similarly, $V^n(x)$ can be expressed recursively as the sum of the expected reward received after one step plus the discounted expected return for an $n-1$ horizon problem that follows. That is,

$$V^n(x) = \max_{a \in \mathbf{A}} [\rho(x, a) + \gamma \mathbf{E}[V^{n-1}(T(x, a))], \tag{3.27}$$

where

$$\mathbf{E}[V^{n-1}(T(x, a))] = \sum_{y \in \mathbf{S}} P_{xy}(a) V^{n-1}]. \tag{3.28}$$

The optimal policy for a finite horizon problem is obtained by choosing the action that achieves the maximum in Equation 3.27. If f^n denotes the optimal policy for an n-step finite horizon problem, then

$$f^n(x) = \arg\max_{a \in \mathbf{A}} [\rho(x, a) + \gamma \mathbf{E}[V^{n-1}(T(x, a))]. \tag{3.29}$$

By starting at $n = 0$, V^n can be computed by repeatedly applying Equation 3.27. For finite rewards and $\gamma < 1$ as $n \to \infty$, $|V^n - V^*| \to 0$. Similarly, since V^n converges to V^*, f^n converges to f^* (see Ross for further details [Ross, 1983]).

The value iteration algorithm is summarized in Figure 3.5. Notice that each stage of the process is computationally equivalent to calculating the action-value function in policy iteration; however, in value-iteration the value function for the next stage is determined automatically and we don't have to compute it by solving a set of linear equations.

```
$V^0(x) = 0$ for all $x$,
$Q^0(x,a) = 0$ for all $(x,a) \in \mathbf{S} \times \mathbf{A}$
$n = 0$,
Repeat
  $n = n + 1$,
  for each $x \in \mathbf{S}$ do:
    for each $a \in \mathbf{A}$ do:
      $Q^n(x,a) = \rho(x,a) + \gamma \sum_{y \in \mathbf{S}} P_{xy}(a) V^{n-1}(y)$,
    $V^n(x) = \max_{a \in \mathbf{A}} Q^n(x,a)$
    $f^n(x) = \arg\max_{a \in \mathbf{A}} Q^n(x,a)$
until the difference between $V^n$ and $V^{n-1}$ is sufficiently small for all $x$.
```

Figure 3.5: The Value Iteration Algorithm.

3.3 Q-Learning

There is a close relationship between reinforcement learning and dynamic programming. In both, the world is characterized by a set of states, a set of actions, and a reward function. In both, the objective is to find a decision policy that maximizes the expected cumulative reward received over time. There is an important difference though. When solving a Markov decision problem with dynamic programming, the analyst (presumably the designer of the eventual control system) has a complete (albeit stochastic) model of the environment's behavior. Given this information, the analyst directly *computes* the optimal policy. In reinforcement learning, the set of states and the set of possible actions are known *a priori*, but the effects of actions and the production of reward are initially *unknown*. Thus, an analyst cannot compute an optimal policy *a priori*. Instead, an optimal control strategy must be learned through experimentation in the environment. Dynamic programming is an offline analytical tool; reinforcement learning is an online adaptive control technique.

The work in this dissertation is based on a reinforcement learning algorithm called Q-learning developed by Chris Watkins [Watkins, 1989]. We focus on Q-learning because of its simplicity and because of its close relationship with dynamic programming and the Theory of Markov decision processes. Q-learning is also significant because one version of it, namely 1-step Q-learning, when applied to

Markov decision processes under appropriate conditions, can be shown to converge to an optimal policy. Although there are several varieties of reinforcement learning algorithms, most are based on ideas from dynamic programming and are quite similar to Q-learning [Samuel, 1963; Michie and Chambers, 1968; Sutton, 1984; Holland, 1975; Kaelbling, 1990].

The connection between Q-learning and dynamic programming is strong. Q-learning can be thought of as a form of incremental dynamic programming, where the optimal action-value function is computed incrementally based on the system's interactions with its environment. Q-learning can be derived as a modification of either policy iteration or value iteration. The following section derives Q-learning as a form of incremental policy iteration.

3.3.1 Q-Learning as Incremental Policy Iteration

Recall that in policy iteration, an optimal policy is obtained upon convergence of a sequence of policy improvement stages. During each stage, the action-value function, Q_f, for the current policy is computed and used to derive an improved policy for the next stage (see Figure 3.4). The improved policy, f', is chosen by selecting for each state the action that locally maximizes the expected return. That is,

$$\forall_{x\in \mathbf{S}} f'(x) = a \text{ such that } Q_f(x,a) = \max_{b\in \mathbf{A}} Q_f(x,b). \tag{3.30}$$

The new policy (via the policy improvement theorem) is guaranteed to be uniformly as good as or better than the current policy.

Computing the action-value function is a crucial step in policy iteration. Classically, Q_f is determined by first solving for V_f (using the linear equations defined in Equations 3.11 and 3.12) and then using Equation 3.13 to compute its values. Unfortunately, both of these steps require prior knowledge of the statistics that govern the process (namely $\rho(x,a)$ and $P_{xy}(a)$), which for learning tasks are initially unknown.

If a learning algorithm based on policy iteration is to be developed, it will be necessary to find a way to compute the action-value function. One way to proceed is to estimate the needed statistics directly. Following this approach one can use observations of interactions with the world to develop estimates of $\{\rho(x,a)\}$ and $\{P_{xy}(a)\}$. These estimates can then be used to compute *estimates* of the action-value functions and eventually compute an *estimate* of the optimal policy. If the world is stationary, and if each state-action pair is tried often enough, then accurate estimates can be obtained and eventually the optimal policy learned.

The trouble with this approach is that it is not particularly incremental. It is too expensive to recompute the policy after each step. Indeed, computing a policy just once can be very expensive, especially for large problems. If this method were to be used, it would be necessary to update the policy periodically, where the recomputation rate would depend on the time required to do it. During the interval between recomputations the agent's policy would be frozen and it would not be able to take advantage of the experience it had gained since the last policy computation. Also, this method would be inappropriate if the agent had to act continuously in the world. An agent may not be able to afford to take time out to recompute its policy.

A better approach that is computationally less expensive and more incremental is to appeal directly to the definitions of V_f and Q_f and estimate them directly via Monte Carlo simulation. One way to estimate $V_f(x)$ directly is simply to average the returns observed over multiple trials (or sequences) that begin in state x and follow f. A similar technique can be used to estimate Q_f.

Estimating V_f and Q_f

If the system is in state x at time t, then an estimate of $V_f(x)$ can be obtained by observing the rewards obtained by following f. The total discounted reward received in such a sequence is called the *actual return* for time t and is given by:

$$\mathbf{r}_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^n r_{t+n} + \ldots. \qquad (3.31)$$

After n steps an estimate of the actual return can be obtained by considering rewards received so far. This is called the *n-step truncated return*,

$$\mathbf{r}_t^{[n]} = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^n r_{t+n}. \qquad (3.32)$$

If rewards are bounded and $\gamma < 1$, then $\lim_{n\to\infty} |\mathbf{r}_t - \mathbf{r}_t^{[n]}| = 0$. So for a sufficiently large n an accurate estimate of the actual return can be obtained.

An accurate estimate of $V_f(x)$ can then be obtained by averaging a large number of n-step truncated returns. An estimate of V_f can then be obtained by performing enough of these experiments for each state-action pair.

The advantage of estimating the value function directly using Monte Carlo methods, instead of computing it by solving a set of linear equations, is that the estimate can be updated incrementally as new experiences provide additional information. However, there are still a number of problems with this approach:

1. The value of the truncated return is only available after n steps. If n is large, as it should be to obtain an accurate estimate, this delay could be significant. Also, if truncated returns for multiple state-action pairs are to be obtained simultaneously, then a possibly long sequence of state, action, and reward triples must be remembered.

2. For stochastic systems, the truncated returns obtained may have a large variance, and many experiments will be required to estimate the expected value accurately.

3. In order to obtain the truncated return, the agent must follow f for n steps before it can try a non-policy action. Thus, non-optimal actions can only be tried every n steps, and return estimates for non-optimal actions are expensive to obtain.

These problems can be overcome by using a different kind of estimator. Instead of accumulating actual rewards for a long series of steps, we can define an estimator that has a component for the rewards that have actually been received (up to n steps) and a component that estimates the reward to be received in the future (after n steps). That is, suppose the agent performs an experiment as before. Starting in state x it executes f for n steps. An estimate of the actual return received can be obtained by augmenting the n-step truncated return with a term that accounts for the reward that would be received after time $t+n$. If the state obtained after n steps is x_{t+n}, then the expected value of the reward to be received after n steps, assuming the system follows f, is just $V_f(x_{t+n})$. Of course, V_f is not known ahead of time. It is what we are trying to estimate. However, an approximation of V_f can be obtained from our current estimate. Let U_t denote the approximation to V_f at time t.

The *corrected n-step truncated return* is then defined as

$$\mathrm{r}_t^{(n)} \equiv r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \gamma^n U_t(x_{t+n}). \tag{3.33}$$

If U equals V_f, then $\mathrm{r}_t^{(n)}$ is an unbiased estimator of V_f. That is,

$$\mathbf{E}[\mathrm{r}_t^{(n)}] = V_f(x). \tag{3.34}$$

Even when U_t is not an accurate estimate of V_f, the corrected n-step estimate is still useful for estimating the action-value function. This is due to what Watkins calls the *error-reduction property* of corrected truncated returns [Watkins, 1989]. In particular, let M be the maximum absolute error in U_t with respect to V_f, that is,

$$M = \max_{x \in \mathbf{S}} |U_t(x) - V_f(x)|. \tag{3.35}$$

Then clearly,

$$\max_{x \in \mathbf{S}} |\mathbf{E}[\mathrm{r}_t^{(n)}(x)] - V_f(x)| \leq \gamma^n M. \tag{3.36}$$

The significance of this result is that the average of the corrected truncated returns for any given state will never have an error greater than $\gamma^n M$. Also, M will tend to zero as experiments with the maximally erroneous state accumulate. Thus,

for a fixed policy f, the value function can be estimated accurately by averaging corrected n step truncated returns.

There are two key advantages to corrected truncated returns. First they are effective even when n is small. The error-reduction property holds for all $n > 0$. This means that 1) estimates for the action-value function can be obtained without excessive delay, 2) estimates for non-policy actions can be obtained more easily since the agent need not follow its policy for a long sequence of actions in order to obtain useful estimates, and 3) variation in the estimate due to the sum of a large number of random variables can be reduced. In general, there is a tradeoff between the variance in an estimate and the effect of bias in an estimate due to error in U_t. For large n, the variance in a single estimate will be large compared to that for small n. Conversely, the impact of errors in U_t decreases exponentially with increasing n.

A second advantage of corrected truncated returns is that a weighted sum of corrected truncated returns for different values of n can be implemented efficiently and locally in time. In particular, the $n+1$-step estimator can be expressed in terms of the n-step estimator as follows:

$$r_t^{[n+1]} = r_t^{[n]} + \gamma^n[r_{t+n} + \gamma U_t(x_{t+n+1}) - U_t(x_{t+n})]. \tag{3.37}$$

The bracketed term on the right side of the equation can be computed locally in time since it depends only upon the observables r_t, x_{t+n+1} and x_{t+n}. Using this relationship, a series (or weighted sum) of corrected estimators can be efficiently computed over time and used to update estimates of V_f and Q_f incrementally. The bracketed term on the right is the difference between two estimates of $V_f(x_{t+n})$. The term $U_{t+n}(x_{t+n})$ is an estimate available at time $t+n$. The term $r_{t+n} + \gamma U_{t+n+1}(x_{t+n+1})$ is an estimate obtained by waiting for one step and observing the next state and rewards that follow. The term $[r_{t+n} + \gamma U_t(x_{t+n+1}) - U_t(x_{t+n})]$, called the *temporal difference error*, estimates the error in $U_{t+n}(x_{t+n})$ based on information gained after one time step. Techniques based on these temporal differences (TD) were first developed and formally analyzed by Sutton, who showed them to be useful for a wide range of prediction and estimation tasks [Sutton, 1984; Sutton, 1988; Sutton and Pinette, 1985].

Incremental Policy Improvement

Equipped with a range of methods for estimating an agent's action-value function, we must now consider the second major operation in the policy iteration algorithm, policy improvement. A straightforward approach is to update the policy only after an accurate estimate of Q_f has been obtained. The trouble with this approach is that it may take a long time to estimate Q_f. In the meantime, the agent's policy will continue to be suboptimal and fail to reflect the information gained since the last updating. In Q-learning, the agent's policy is updated after each time step so that it reflects the most current estimate of the action-value function and the experience gained so far.

A particularly simple version of the Q-learning algorithm is shown in Figure 3.6 The first step in this algorithm is to initialize the agent's action-value function. If prior knowledge of the task is known, it may be possible to choose initial values that positively bias the action-value function. Otherwise, random or uniform initial values will do. Next, the agent enters the main control cycle. The first step in the control cycle is to determine the current state of the world.[2] Next, the agent evaluates its policy function for the current state to obtain an estimate of the optimal control action. Most of the time the agent executes this action, but occasionally an action is chosen at random. Random actions help to ensure experimentation with all state-action pairs. The general question of how to trade off exploration (acting to gain information or experience) against exploitation (acting to gain reward) is difficult to answer and has a long history in game theory, matł ematical statistics, and optimal control [Robbins, 1952; Bradt *et al.*, 1956; Bradt and Karlin, 1956; Feldaman, 1962; Dubins and Savage, 1965; Epstein, 1967; Holland, 1973; Holland, 1975; Kaelbling, 1990; Hartman, 1990]. The algorithm in Figure 3.6 takes a particularly simple approach, choosing on each step a random action with a fixed probability p. A slightly more sophisticated approach is to choose actions according to a Boltzmann distribution, where the probability of selecting action a in state x is given by

$$p(a|x) = \frac{e^{Q(x,a)/T}}{\sum_{b \in A} e^{Q(x,b)/T}} \tag{3.38}$$

and where the temperature parameter T is slowly decreased over time in order to decrease exploration as experience accumulates. After performing the selected action, the agent observes the next state, y, and the reward, r, obtained. These observations are used to construct a corrected 1-step estimate for $Q(x_t, a_t)$,

$$r^{(1)} = r + \gamma U(y). \tag{3.39}$$

The 1-step estimator is particularly convenient since it is immediately available. However, more elaborate multi-step estimators can also be used. Next (Step 5), the action-value for the current state-action pair is updated using the rule:

$$Q(x,a) := (1-\alpha)Q(x,a) + \alpha[r + \gamma U(y)] \tag{3.40}$$

where α is a learning rate parameter ranging between 0 and 1. This updating rule implements a temporally weighted average, where the recent estimates receive

[2]Typically, it is assumed that the state is defined in terms of the agent's current sensory inputs However, this is not an intrinsic part of the theory.

```
$Q \leftarrow$ a set of initial values for the action-value function (e.g., uniformly zero)
$f(x) := a$ such that $Q(x,a) = \max_{b \in A} Q(x,b)$,
Repeat forever:
  1) $x :=$ the current state
  3) Select an action $a$ to execute that is usually consistent with $f$
     but occasionally an alternate. For example, one might choose to
     follow $f$ with probability $p$ and choose a random action otherwise.
  4) Execute action $a$, and let $y$ be the next state and $r$ be the reward
     received.
  5) Update $Q(x,a)$, the action-value estimate for the state-action pair $(x,a)$:
     $Q(x,a) \leftarrow Q(x,a) + \alpha[r + \gamma U(y)]$
       where $U(y) = Q(y, f(y))$.
  6) Update the policy $f$:
     $f(x) = a$ such that $Q(x,a) = \max_{b \in b f A} Q(x,b)$,
```

Figure 3.6: A simple version of the 1-step Q-learning algorithm.

more weight than old estimates. Temporally sliding estimates of this type are appropriate for adaptive control in general since the task may be non-stationary. However, they are especially appropriate for Q-learning since early action-value estimates may have large errors compared to later ones. Finally (Step 6), the agent's policy is updated and a new cycle begins.

3.3.2 The Convergence of Q-Learning

Even though Q-learning is based on dynamic programming, it is not clear that it will necessarily learn an optimal policy. Indeed, in the general case (i.e., when using multi-step estimators), the convergence question remains open. However, it has been shown that under appropriate conditions *1-step Q-learning* is guaranteed to converge on an optimal policy [Watkins and Dayan, 1992]. A set of conditions *sufficient* to guarantee convergence to an optimal policy follows:

1. the underlying decision process is Markovian,

2. the corrected 1-step estimator is used to update Q,

3. each state action-pair is tried infinitely often,
4. the learning rate parameter α is varied with time so that α_n, the learning rate at time n,
   (a) is positive and monotonically decreasing to zero,
   (b) the sum of α_n as $n \rightarrow \infty$ diverges, and
   (c) the sum of $[\alpha_n]^2$ as $n \rightarrow \infty$ is finite.

4 The Block Stacking Testbed

We are now in a position to address the central topic of this dissertation, namely, the integration of active perception (for efficient task-oriented perception) and reinforcement learning (for adaptive, non-model based control). This chapter describes the specific block manipulation task used in the experiments and also presents a formal mathematical model for describing the control tasks facing an embedded learning system that interacts with the world through an active visuomotor system.

4.1 The Block Manipulation Task

The external part of the block manipulation task is quite similar to the block stacking tasks studied in classical planning [Nilsson, 1980; Sussman, 1975]. The block manipulation task involves a simulated robot working on a simulated assembly line, as shown in Figure 4.1. The robot's job is to arrange piles of blocks into certain desirable (or goal) configurations before they fall off the end of a conveyor. If the robot manages to configure the blocks properly, it receives a positive reward, the pile disappears, and a new pile appears at the head of the conveyor. If after n_{quit} steps the robot fails to configure the pile, the blocks fall off the conveyor, the trial ends, and a new pile appears at the head of the conveyor. The piles coming down the conveyor vary in the number of blocks they contain, in the properties of the blocks, and in the initial arrangement of the blocks. However, only one pile is on the conveyor at a time, and it is possible to properly configure each pile in the allotted time. In general, there are no restrictions on the number of blocks that can be in a pile; however in our experiments we limited piles to 20 or fewer blocks.

The robot's objective is to accumulate as much reward as possible, or in objective terms, to configure piles as quickly as possible.

Blocks have various features that differentiate them from one another and that are used to define "proper configurations." For example, one can imagine blocks

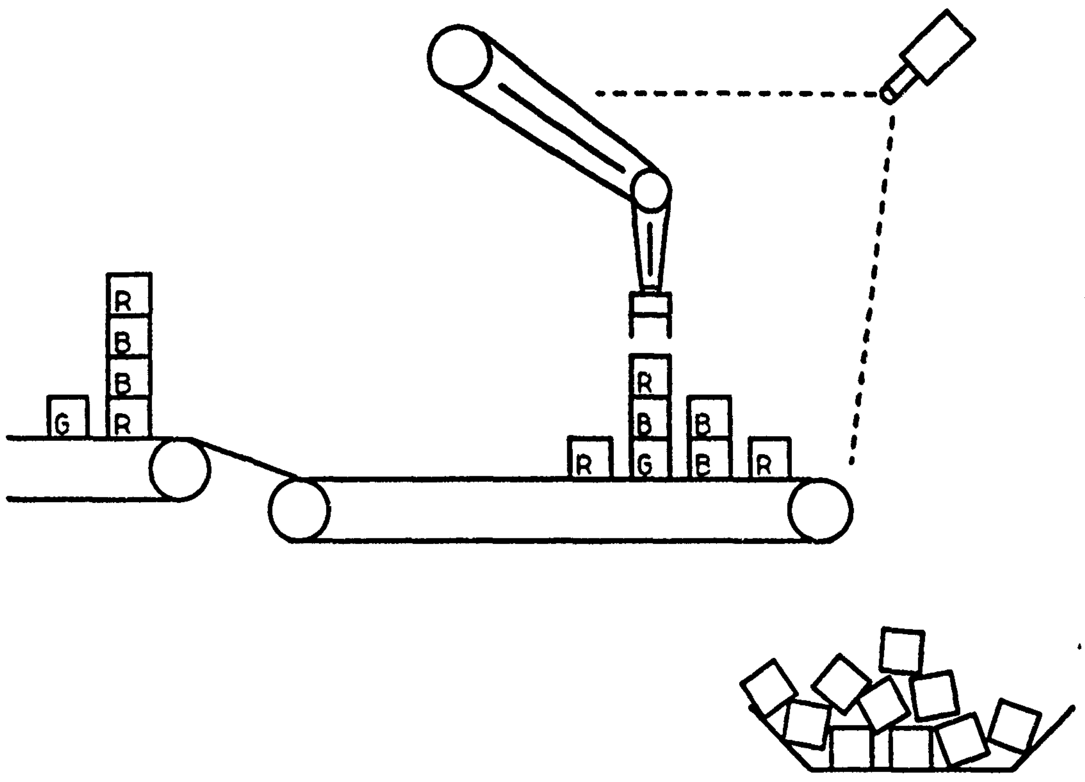


Figure 4.1: A sketch of the block manipulation task. Key properties of this task are 1) the robot has a limited amount of time to configure a pile of blocks properly, 2) piles vary considerably from trial to trial and may contain an arbitrary number of blocks, 3) the robot has no *a priori* knowledge of the task, but learns it by receiving a reward upon the successful completion of a trial, and 4) in addition to learning overt physical actions, the robot must also learn to control an active, but limited, sensory system.

coming in three sizes (small, medium, and large) and three colors (red, green, and blue) and a "proper configuration" being defined as *a-small-green-block-on-a-large-red-block.*

We assume that block dynamics are similar to those typically used in classical planning. That is,

1. The conveyor (or table) is large enough to support an infinite number of blocks.

2. A pile consists of a series of stacks, where each stack is defined to be a column of one or more blocks, and each block can directly support or be directly supported by at most one block.

3. The robot has a single manipulator, which it can use to move one block at a time.

4. Only the robot moves blocks, and the robot's actions have deterministic effects.[1]

5. A block can be picked up only if it does not support another block, and a block can be placed only on the table or on a clear block. Whole stacks cannot be moved, nor can blocks be fitted into the middle of a stack.

Attention is focused on one particularly simple block manipulation task, which we will call the GB-task (for GREEN-BLOCK-task). In this task, blocks come in three colors: red, green, and blue, and the agent receives a positive reward, R_g, whenever it manages to pick up a green block. That is, goal states consist of those configurations of the physical world in which the robot is holding a green block. We also assume that each pile contains exactly one green block.[2] Even though this task may seem simple, it is important to realize that it is not completely trivial since the robot may need to unstack a considerable number of blocks before it can uncover the green block. Also, since the robot is to learn the task, its sensor and effectors are initially uninterpreted and it has no *a priori* knowledge of the task. Finally, keep in mind that the robot has a sensory-motor system that is flexible, but limited in the amount of information it provides the decision system.

[1]As we will see in later chapters, for certain instantiations of the CR-method this assumption can be relaxed.

[2]The reason for this assumption will be made clear in the coming chapters, when we describe an approach for dealing with multiple green blocks. In the meantime, however, the intuitive reason for the restriction is to simplify the amount of sensory processing required by the robot. In particular, if multiple green blocks exist, then the robot (if it is to behave optimally) must analyze and compare the utility of strategies for pursuing each green block. While this selection process is important, it is a complication that gets in the way of studying more basic issues, so it is ignored for now.

In addition to learning to perform the correct overt actions, the robot must also learn to control its sensory system in order to generate an adequate, task-specific representation.

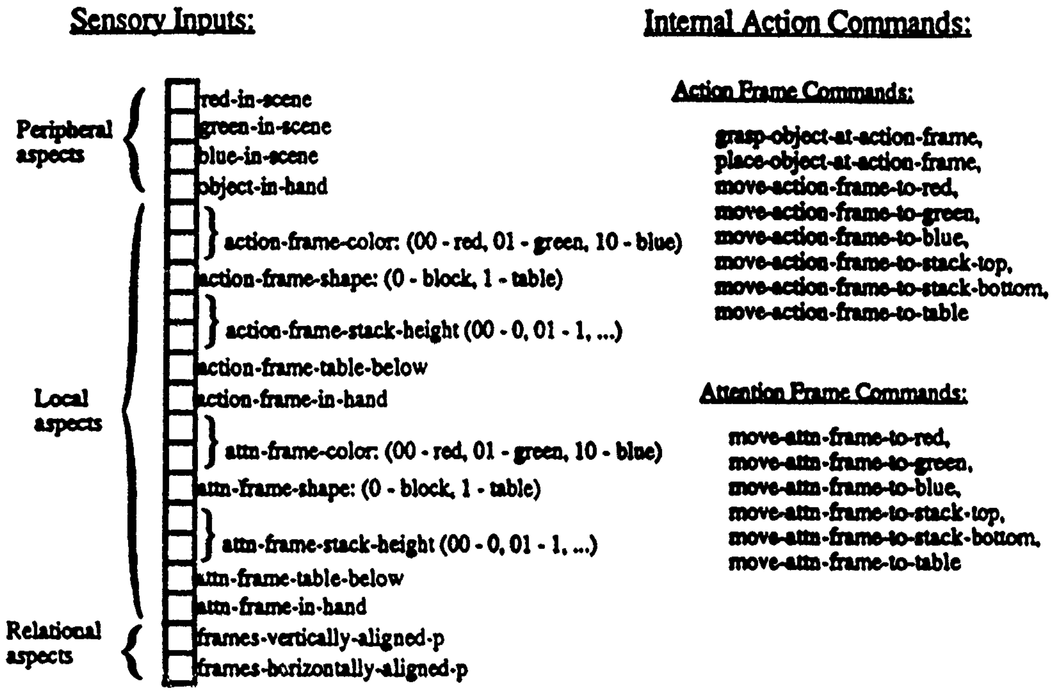
4.2 The Sensory-Motor System

The sensory-motor system is considerably different from the objective representations traditionally used. The robot is equipped with an active sensory-motor system that provides selective, but limited, access to the external world. Our contention is that many problems of interest only require keeping track of a few objects at a time (for example, see [Chapman, 1989]).

The specification for the embedded decision system's interface to the sensory-motor system is shown in Figure 4.2. On the sensory side, the system at each point in time generates a 20-bit binary input vector, which defines the state of the agent's internal representation. The information encoded in the input vector falls into three general categories: peripheral aspects, local aspects, and relational aspects. In general, *peripheral aspects* register spatially non-specific information about the world, such as the presence or absence of indexical properties (e.g., colors, shapes, and textures). Our robot's sensory-motor system has 3 peripheral bits that register the presence of red, green, and blue in the scene, and 1 bit that registers whether or not the robot is holding an object. Both local and relational aspects register properties of objects that are marked.[3] *Local aspects* register intrinsic local features of an object, such as its shape, color, orientation, and texture. Relational aspects register relational properties between marked objects, such as relative shape, relative color, and relative position. The system in our experiments has two markers, called the *action-frame* and the *attention-frame*, respectively. The local aspects for these markers define 14 of the 20 bits in the input vector. Local aspects register the color of the marked object (2-bits/marker), the shape of the object (1-bit/marker), whether or not the marked object is currently being held (1-bit/marker) and the number of blocks on top of the marked object (2-bits/marker). The robot's sensory system registers two relational aspects — one for recording vertical alignment between markers (1-bit) and one for recording horizontal alignment (1-bit).

The internal motor commands supported by the sensory-motor systems are shown on the right in Figure 4.2. Two types of internal motor commands are distinguished: overt actions and perceptual actions. *Overt actions* are commands that change the state of the external world with respect to the task. This can either occur by performing a physical action that manipulates some object in the

[3]These bits correspond to aspects in Agre and Chapman's theory [Agre, 1988; Chapman, 1990b]



Sensory Inputs:
Internal Action Commands:
Peripheral aspects
red-in-scene
green-in-scene
blue-in-scene
object-in-hand
Local aspects
action-frame-color: (00 - red, 01 - green, 10 - blue)
action-frame-shape: (0 - block, 1 - table)
action-frame-stack-height (00 - 0, 01 - 1, ...)
action-frame-table-below
action-frame-in-hand
attn-frame-color: (00 - red, 01 - green, 10 - blue)
attn-frame-shape: (0 - block, 1 - table)
attn-frame-stack-height (00 - 0, 01 - 1, ...)
attn-frame-table-below
attn-frame-in-hand
Relational aspects
frames-vertically-aligned-p
frames-horizontally-aligned-p
Action Frame Commands:
grasp-object-at-action-frame,
place-object-at-action-frame,
move-action-frame-to-red,
move-action-frame-to-green,
move-action-frame-to-blue,
move-action-frame-to-stack-top,
move-action-frame-to-stack-bottom,
move-action-frame-to-table
Attention Frame Commands:
move-attn-frame-to-red,
move-attn-frame-to-green,
move-attn-frame-to-blue,
move-attn-frame-to-stack-top,
move-attn-frame-to-stack-bottom,
move-attn-frame-to-table

Figure 4.2: A specification for the block stacking robot's sensory-motor system. The system has two markers, an *action-frame* marker and an *attention-frame* marker. The system has a 20-bit input vector, 8 overt actions, and 6 perceptual actions. The values registered in the input vector and the effects of internal action commands depend upon the bindings between markers in the sensory-motor system and objects in the external world.

physical world (e.g., picking up a block) or by changing the configuration in the sensory-motor system in a way that changes the robot's ability to manipulate objects in the external world (e.g., moving a marker that establishes the reference frame used by other overt actions). *Perceptual actions* are commands that change the agent's perception of the world but do not affect the external world or the agent's ability to interact with it. The robot has a total of 14 internal motor commands: 8 are overt, 6 are perceptual.

All overt actions are associated with the action-frame marker. In addition to providing perceptual information, this marker is also used to establish the reference frames for manipulating objects in the world. The two primary overt actions are for grasping and for placing objects. For grasping, the action *grasp-object-at-action-frame* causes the robot to pick up the object marked by the action-frame marker. The action succeeds if the robot's hand is empty and the marked object has a clear top. For placing, the action *place-object-at-action-frame* causes the system to place a block it is holding on top of the object pointed to by the action-frame marker. This action works if the robot is holding a block and the target object has a clear top. The remaining overt actions are used to move the action-frame to different spatial locations. These commands allow the agent to index functionally relevant objects by primitive indexical properties (color and shape) or by spatial relationships (top-of-stack and bottom-of-stack). Although these indexing actions do not change the external world directly (i.e., no blocks get moved), they are overt actions in the strictest sense because they affect the robot's ability to manipulate the external world. Moving the action frame changes the effects of grasp and place actions. Once a marker is placed on an object, it tracks the object until the binding is explicitly broken with another indexing command.

A repertoire of perceptual actions are associated with the attention-frame marker. These actions are used exclusively for gathering additional sensory information. No overt actions depend on the placement of the attention-frame marker. As for the action-frame, indexical properties are used to guide the placement of the attention frame. As will be seen in Chapter 6, the attention frame marker plays an important role in allowing the system to disambiguate functionally different situations in the world.

The robot's sensory-motor system was directly motivated by Ullman's visual routines model and Agre and Chapman's work on deictic representations; and although it is a simplification, it embodies many of the essential ideas of the active-perception paradigm. In particular, the agent's internal representation is flexible, but limited in scope. This follows since the internal representation is almost completely defined in terms of the action and attention frame markers (or processing foci), which are actively controlled by a higher level decision component. One feature that may appear to be missing is the ability to assemble complex visual routines from elemental operations. However, this is not the case since in general visual routines are assembled and dispatched by higher level decision components. In our robot, "visual routines" emerge as the decision system learns to control the sensory-motor system in order to gain needed information. Admittedly, some of our "primitive aspects" are unrealistically complex and would probably be implemented using visual routines in a more realistic system.

Notice that the internal state space defined by the sensory inputs is small compared to the state space that could result if every object in the domain were represented objectively [Swain, 1990]. The principal advantage of this reduced internal representation is that it leads to more feasible perception and a simpler decision task. For example, a pile of 50 blocks would require 100 bits and a state space of at least 3^{50} (or 7.1×10^{23} states) just to encode the color of each block. Our robot's input vector is limited to 20 bits (or about a million states). The principal disadvantage of the reduced representation is that it limits the complexity of the problems that can be solved by the agent. For example, if during the course of a problem, a decision depends upon features of three separate blocks, then the robot will not be capable of solving the problem since it cannot simultaneously represent features of more than two blocks. Of course, some sort of memory mechanism could be added or the sensory-motor system could be expanded to allow the system to register more information (for example, by adding an additional marker), but in general new problems can always be defined that are beyond the scope of the internal representation.

Also, notice that individual objects in the world are referenced not by arbitrarily assigned names, but by the features that make them relevant. For example, the action *Move-action-marker-to-stack-top* would cause the action-frame to move upwards from its current position until it reaches the block at the top of the stack. What makes this top block significant is not any absolute name like "BLOCK-43," but the relationship it holds with the rest of the world. Namely, this block is at the top of a stack and affords [Gibson, 1979] being removed and placed on the table (possibly to get at another more important block) [Agre, 1988]. The variety of features and properties that can be used as indexicals also delimits the types of problems that an agent can solve.

Finally, notice that physical actions in the world (e.g., picking and placing blocks) are performed relative to reference frames established by the action-frame. This is consistent with the view that objects in the world fill roles according to their features and that the control strategy learned by the decision system should be specified in terms of those abstract roles [Agre, 1988; Ballard, 1989b].

Our objective is to develop adaptive control systems that can learn to solve the GB-task. This objective raises an interesting question. Does such a control strategy exist? That is, given the limited capabilities of the robot's sensory system, does a stationary decision policy (i.e., a fixed mapping from internal states to action commands) exist that solves the task? For the GB-task, the answer to this question is "yes". Figure 4.3 shows a list of condition-action rules that define

If 1) the hand is not empty and
2) the action-frame is not on the table,
then move the action frame to the table.

If 1) the hand is not empty and
2) the action-frame is on the table,
then place the held object at the location marked by the action frame.

If 1) the hand is empty and
2) the attention-frame is not on a green block,
then move the attention-frame to the green block.

If 1) the hand is empty,
2) the attention-frame is on a green block, and
3) the attention-frame and the action-frame are not vertically aligned,
then move the action-frame to the green block.

If 1) the hand is empty,
2) the attention-frame is on a green block,
3) the attention-frame and the action-frame are vertically aligned, and
4) the object marked by the action-frame has a clear top,
then pick up the object marked by the action-frame.

If 1) the hand is empty,
2) the attention-frame is on a green block,
3) the attention-frame and the action-frame are vertically aligned, and
4) the object marked by the action-frame is not clear,
then move the action-frame to the top of the stack.

Figure 4.3: A fixed set of rules that reliably solve the GB-task. Depending upon the distribution of problem instances (piles), this strategy may or may not be optimal. In any case, it is nearly optimal.

such a policy. The GB-task can be solved by 1) keeping the robot's hand clear — so that it can pick up blocks as needed, 2) using the attention-frame to locate the green block, 3) when not involved in placing an object, move the action-frame to the top of the stack containing the green block, and 4) pick up any clear block from the green stack — either to decrease the stack height or to pick up the green block.

4.3 Perceptual Mappings

Let us study the properties of an active perceptual system in more detail. Perception can be defined as the process of mapping situations in the world onto states in an agent's internal representation. Following this definition in the most general sense, the perceptual system can include all the sensory and computational processes that provide information to the internal representation. This could include short term memory processes used to maintain and recall information about previous events. However, we will restrict ourselves to agents whose internal representations are defined solely in terms of their immediate sensory inputs. In this case, the agent's sensory system performs the mapping from situations in the world to internal states.

Since in general there are an unbounded number of different situations in the world (i.e., every moment is unique in some respect), and we are concerned with systems that have only a finite number of internal states, some internal states must necessarily represent multiple world states. We call this overloading of internal states *perceptual reduction*. Perceptual reduction is fundamental to perception, it cannot be avoided. However, it can be helpful or it can be a hindrance. In particular, if the perceptual mapping is chosen correctly, then each internal state will represent situations that are *functionally equivalent*. Conversely, if the mapping is chosen incorrectly then the equivalence class associated with some internal states may contain situations that are functionally dissimilar. Under these circumstances, the internal state may say nothing useful about the current situation with respect to the task. Agre has called the correct or useful overloading of internal states *passive abstraction* [Agre, 1988]. We will adopt this nomenclature and use the term *perceptual aliasing* to denote the incorrect (or unproductive) overloading of internal states.

An example of passive abstraction for the GB-task is shown in Figure 4.4. In this case, two different piles of blocks in the world, due to careful placement of the action and attention frames, generate the same internal representation. The equivalence class associated with this internal state consists of those situations where:

1. the hand is empty.

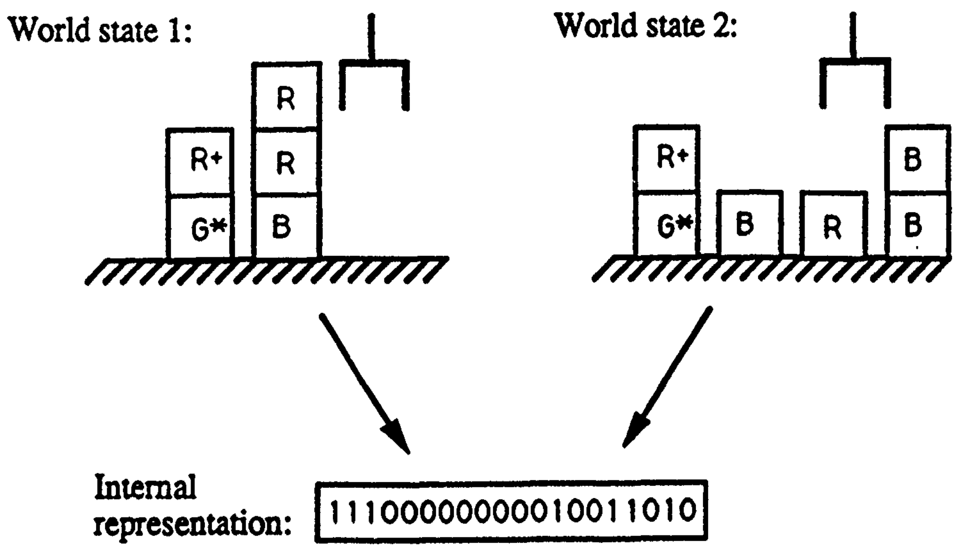


Figure 4.4: An example of passive abstraction in the GB-task. In this case, both world states share the same optimal action. In the figure the (+) represents the location of the action-frame marker and the (*) represents the location of the attention-frame marker.

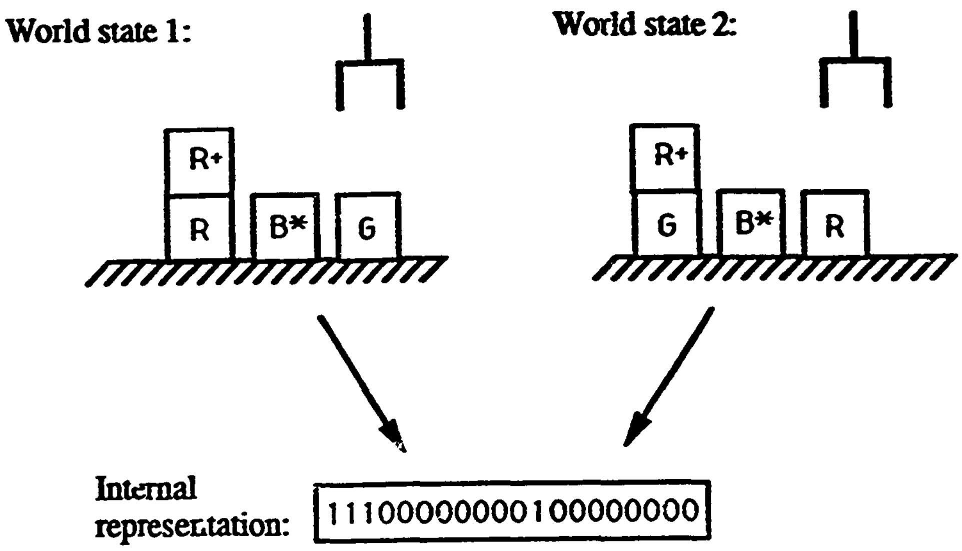
2. a green block is covered by a red block that itself is clear (there may be other blocks between the red and the green ones),
3. the action-frame marker is on the red block,
4. there are red, green, and blue blocks in the pile.

With respect to the GB-task, whenever this internal state is encountered the optimal action to perform is *grasp-object-at-action-frame.* That is, situations represented by this particular sensory input vector are functionally equivalent with respect to the the GB-task.

Notice how information about the other irrelevant blocks in the pile is not encoded in the internal representation. Most of the irrelevant information (e.g., information about stacks other than the one containing the green block) is abstracted out of the representation automatically. Nevertheless, this internal state does encode some irrelevant information — in particular, the fact that the top block is red and that there are blue, green, and red colors detected in the scene is irrelevant.

Intuitively, an internal state is most useful when

1. *the equivalence class associated with the state is as large as possible (i.e., the representation does not make irrelevant distinctions), and*



Figuie 4.5: An example of perceptual aliasing from the GB-task. In this case two world states with different optimal actions generate the same internal representation.

2. *all the situations i.: the world represente·· y a state are functionally equivalent in terms of the actions required for optimal control.*

An example of perceptual aliasing is shown in Figure 4.5. In this case, marker placements are such that two functionally different situations generate the same input vector. The optimal action for the pile on the left is to move the action-frame to the green block (*move-action-fiame-to-green*), whereas, the optimal action for the pile on the right is to pick up the marked red block (*grasp-object-at-action-frame*). The trouble with this internal state is that, given only this information, the decision system cannot distinguish between these two different cases and so cannot be guaranteed to select the optimal action.

In systems with fixed (or passive) sensory systems, the burden of choosing an appropriate internal representation (and perceptual function) is placed on the system's designer. The objective representations commonly used in AI are passive representations. In this case, every potentially relevant object is identified, named, and objectively represented. Also, they are typically intended to be general purpose so that the robot can perform a range of tasks in the domain. Objective representations tend to be bad representations, not so much because they confuse situations that are functionally different, but because they differentiate between situations that are functionally equivalent. In objective representations, each internal state encodes too much information for most tasks. This complicates decision making by forcing the agent to do its own abstraction.

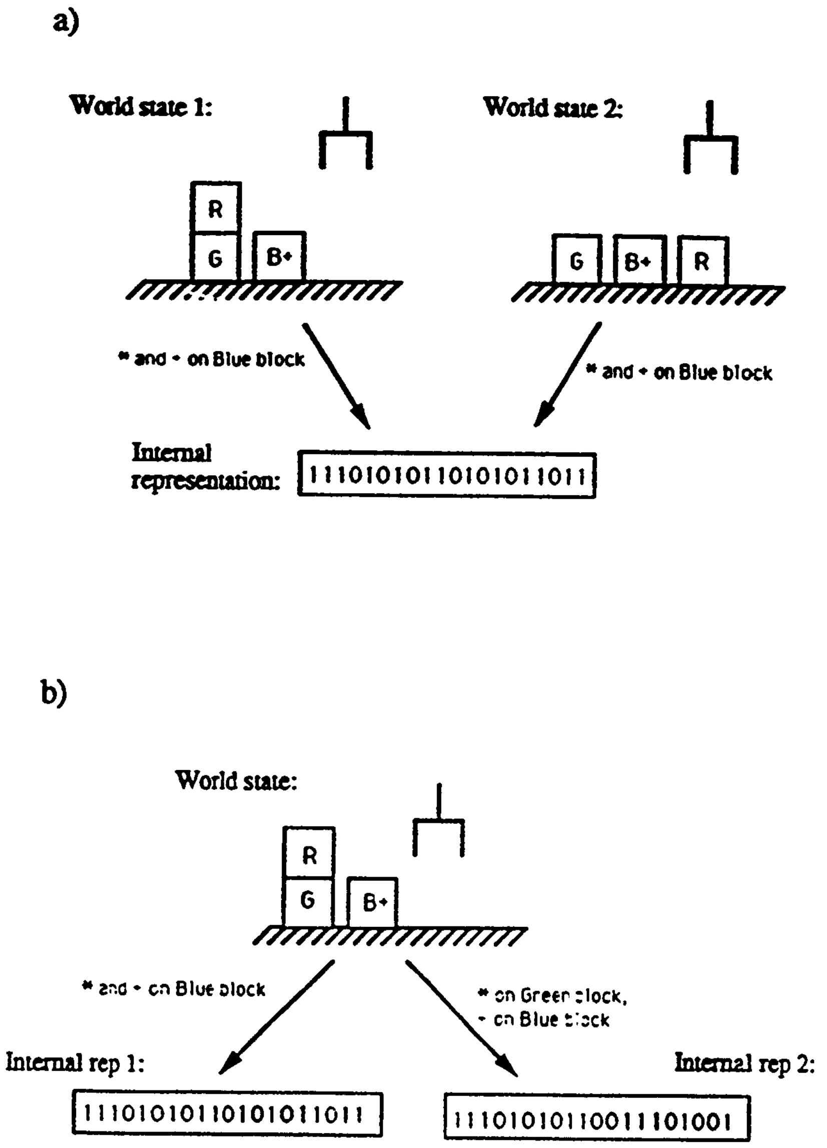


Figure 4.6: Generally the mapping between external world states and the agent's internal representation is many-to-many; a) shows how two different situations can generate the same internal state and b) shows how one situation may have more than one internal representation.

To obtain a more appropriate representation, it is necessary to consider the specific task to be performed by the robot and the functional equivalence classes it entails. This can be a problem if the task is initially unknown or changes over time. However, active perception can address this problem by providing an efficient means for implementing a wide range of perceptual mappings. Of course, in the case of active perception, adaptive control involves the learning of both the overt actions needed to perform the task and the perceptual actions needed to sense and represent the world properly with respect to the task. Examples of the many-to-many mappings afforded by the block stacking robot's sensory-motor system are shown in Figure 4.6.

4.4 A Formal Model of Embedded Learning

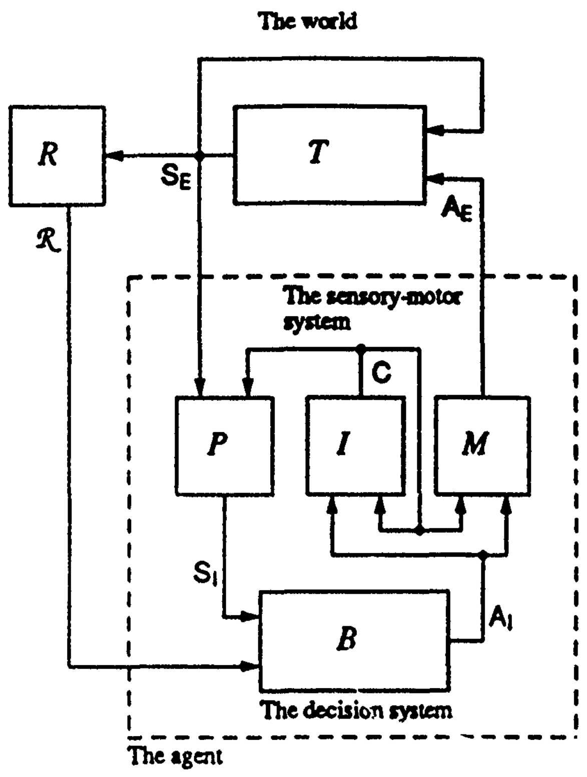
Let us now formalize concepts such as "the world," "the agent," "the sensory-motor system" and the "decision system" in a general model. The model, shown in Figure 4.7, extends a model proposed by Kaelbling [Kaelbling, 1989] by explicitly representing the dynamic relationship between external world states and the agent's internal representation. Given this model we can formally describe the decision problem facing the embedded controller.

4.4.1 The External World

The external world is modeled as a discrete time, discrete state, Markov decision process and is described by the tuple (S_E, A_E, T_E, R_E). The 'E' subscript is used to emphasize that this is a model of the environment external to the agent. The model is a mathematical abstraction of the physical world that collapses the infinite complexity of the real world onto a finite model. However, the model is assumed to capture the essence of the task to be performed by the agent (or put another way, the Markov model defines that task to be performed). Of course, the model is just a mathematical abstraction, and the agent (and especially the embedded decision system) has no knowledge of it.

4.4.2 The Agent

Our model of the agent has two major subsystems: a sensory-motor subsystem and a decision subsystem. The sensory-motor subsystem implements three functions: 1) a perceptual function P; 2) an internal configuration function I; and 3) a motor function M. The model of the sensory-motor system formalizes the relationships that exist between 1) internal states and actions and 2) the external world model. On the sensory side, the system translates world states (i.e., states in the external



Symbol	Function Name	Mapping	Probabilistic	Adaptive
T	Transition	$S_E \times A_E \rightarrow S_E$	Yes	No
R	Reward	$S_E \rightarrow \mathcal{R}$	Yes	No
P	Perceptual	$S_E \times C \rightarrow S_I$	No	No
I	Configuration	$A_I \times C \rightarrow C$	No	No
M	Motor	$A_I \times C \rightarrow A_E$	No	No
B	Behavioral	$(S_I \times \mathcal{R})^* \rightarrow A_I$	Yes	Yes

Figure 4.7: A formal model for an agent with an embedded learning system and an active sensory-motor system. The table summarizes the functions implemented by each of the model's components.

model) into the states in the agent's internal representation. Since perception is active, this mapping is dynamic and dependent upon the configuration of the sensory-motor apparatus. Let S_I be the finite set of possible internal states, and C be the set of sensory-motor configurations. Then, the relationship between world states and the agent's internal representation can be described by a ***perceptual function*** P, which maps world states S_E and sensory-motor configurations C onto internal states S_I (i.e., $P : S_E \times C \rightarrow S_I$). Notice that in a real system, the physical sensory system implements a mapping from the physical world (an infinite number of real world situations) onto the agent's internal representation S_I, but in the model, the perceptual function, for a given configuration, maps states in S_E onto S_I.

On the motor side, the agent has a finite set of *internal motor commands*, A_I, that affect the model in two ways: they can either change the state of the external world (by being translated into external actions, A_E), or they can change the configuration of the sensory-motor subsystem. As with perception, the configuration of the sensory-motor system modulates the effects of internal commands. This dependence is modeled by the functions M and I, which map internal commands and sensory-motor configurations into actions in the external world and into new sensory-motor configurations, respectively (that is, $M : A_I \times C \rightarrow A_E$ and $I : A_I \times C \rightarrow C$). M is called the *motor function* and I is called the *Configuration function.* Internal commands that change the state of the external world or that change the sensory-motor configuration so as to affect the motor mapping are called overt actions and are denoted by the set A_O. Commands that change the configuration of the sensory-motor system, but leave the motor mapping unchanged, are called perceptual actions and are denoted by the set A_P.

The other component of the agent is the decision subsystem. This subsystem is like a homunculus that sits inside the agent's head and controls its actions. On the input side, the decision subsystem has access to reward generated by the external world and to the agent's internal representation, but not to the state of the external world. Similarly, on the motor side, the decision subsystem generates internal action commands that are interpreted by the sensory-motor system.

4.4.3 The Internal Decision Problem

There are two decision problems that need to be distinguished at this point: the external decision problem and the internal decision problem. The external decision problem is defined by the Markov decision process used to objectively describe the external world (or the external task). The internal decision problem is the control task facing the embedded decision system and is defined by the tuple (S_I, A_I, T_I, R_I) where

S_I is the set of possible input values generated by the sensory-motor system,

A_I is the set of internal action commands,

T_I is the internal transition function which describes the effects of internal action commands on the next internal state, and

R_I is the internal reward function.

In this model, the internal transition and reward functions in general depend upon the state of the external world and the configuration of the sensory-motor system. The internal transition function can be expressed in terms of the external transition function, the perceptual function, and the motor function as follows:

$$T_I(x, a|x_E, c) = P(T(x_E, M(a, c)), c) \tag{4.1}$$

where $T(x, a|x_E, c)$ denotes the result of executing internal action command a in internal state x, given that the current world state is x_E and the current sensory-motor configuration is c. Similarly, the internal reward function can be expressed as

$$R_I(x, a|x_E, c) = R_E(x_E, M(a, c)). \tag{4.2}$$

The objective of the embedded decision system is to learn a control policy (a mapping from internal states to internal motor commands) that maximizes the expected future return. Notice, however, that the internal decision problem may or may not satisfy the Markov property since in general transitions and rewards depend upon 1) the state of the external world, 2) the current configuration of the sensory-motor function, and 3) the mappings implemented by the perceptual and motor functions. As will be shown in the next chapter, active perception almost invariably leads to decision problems that are non-Markov.

4.4.4 Modeling the GB-task

The GB-task can be formalized using this model. For the GB-task, the external world can be defined as a Markov process whose state space consists of the set of all possible piles of blocks. In this case, each state describes a unique pile of blocks — much as in an objective representation. The set of actions A_E consists of actions for grasping and placing each and every block. The transition function is deterministic and follows the standard dynamics used in block stacking. The reward function is uniformly zero except for transitions that yield states where the robot was holding a block, in which case the reward is R_g.

The agent in the GB-task is defined as follows. The internal state space S_I is defined by the set of possible values for the input bit vector in Figure 4.2. The set of internal actions A_I are the actions listed on the right in Figure 4.2. The set of configurations C corresponds to the set of possible placements for the markers on objects in the external world. The perceptual and motor functions are somewhat harder to write down, but they define the mapping from piles of blocks in the world to values of the input vector, and from internal action commands to actions in the external "blocks world" model.

5 Combining Active Perception and Q-Learning

The first program written to learn the GB-task, called Meliora-I, takes a most straightforward approach by directly applying 1-step Q-learning to the internal decision problem described in Section 4.4. To our surprise, Meliora-I is unable to learn to solve the GB-task consistently, and generally performs only slightly better than random. In this chapter, we describe Meliora-I, document its poor performance on the GB-task, and analyze its failure. The principal result of the chapter is to show that the agent's that use standard Q-learning (and in general reinforcement learning algorithms that use TD-methods) for the adaptive control of an active sensory-motor system will almost always fail to learn reliable control strategies. The poor performance of Q-learning is explained by introducing the notion of an inconsistent internal state. Informally, inconsistent states are the result of perceptual aliasing and arise any time the perceptual mapping is such that two or more states with differing action-values in the external Markov model get mapped onto the same internal state. Under these circumstances it is not possible for the agent's internal action-value function to represent simultaneously the different action-values for the confounded external states. Consequently, the action-values for inconsistent internal states tend to reflect a *sampled average* of the values for the confounded external states. These averaged values not only lead to inaccurate estimates for the inconsistent states themselves, but also, through the use of TD-methods, introduce errors in the action-value estimates for other internal states. This results in inaccurate action-value estimates for the internal decision problem and non-optimal, indeed unreliable, control policies.

5.1 Meliora-I

Meliora-I uses the 1-step Q-learning algorithm described in Figure 3.6. In this case, the states seen by the controller are the values of the robot's input vector and the set of control actions is the robot's internal motor commands (see Figure 4.2). In total there are 2^{20} (or $1,048,576$) internal states and 14 possible actions. In Meliora-I a table is used to implement the action-value function. The table contains one action-value estimate (an entry) for each state-action pair for a total of 14,680,064 entries. This tabular approach is a particularly simple method for implementing the action-value function; however, its simplicity serves our purposes well. Even though more sophisticated function approximation techniques that save space and provide generalization can be used (e.g., CMAC's, neural networks, classifier systems), they too suffer from the same fundamental difficulties caused by active-perception. Explicit representation of the action-value function in the form of a table makes these interactions more apparent and easier to explain.

5.1.1 The Experiment

The experiment with Meliora-I proceeded as follows. The experiment was comprised of a series of runs, in which the robot was sequentially presented with 1000 instances of the GB-task (i.e., 1000 trials). At the beginning of each run, Meliora-I's action-value function was uniformly initialized to zero. On each time increment, the agent cycled once through the control loop in Figure 3.6. This loop involves choosing and executing an action, observing the state and reward that result, and updating the action-value and policy functions.

Each instance (or trial) consists of a randomly configured pile of exactly 4 blocks with the pile always containing exactly one green block. Randomly selecting problem instances allows the system to get a good mix of easy and difficult problems. An easy problem corresponds to one in which all four blocks are placed on the table. In this case, the robot need merely fixate and directly grasp the green block. A more difficult problem is one in which the green block is at the base of a stack containing all four blocks. In this case, the robot must sequentially unstack all three covering blocks before grasping the green one. If in any trial the robot fails to solve the problem after n_{quit} overt actions, it decides that the instance is too difficult and moves on to the next trial. Limiting the amount of time the robot spends on any given problem instance provides a convenient mechanism for automatically filtering out instances that are far beyond the agent's capabilities at a given point in time. This technique prevents the robot from becoming hopelessly stuck on hard problems during the initial phases of learning.

5.1.2 Results

Because problem instances are selected at random, temporally adjacent trials may vary widely in their difficulty. This results in solution time traces for single runs that appear very noisy. To smooth out the performance curves and make them easier to interpret, the solution time traces for multiple runs are averaged together and plotted.

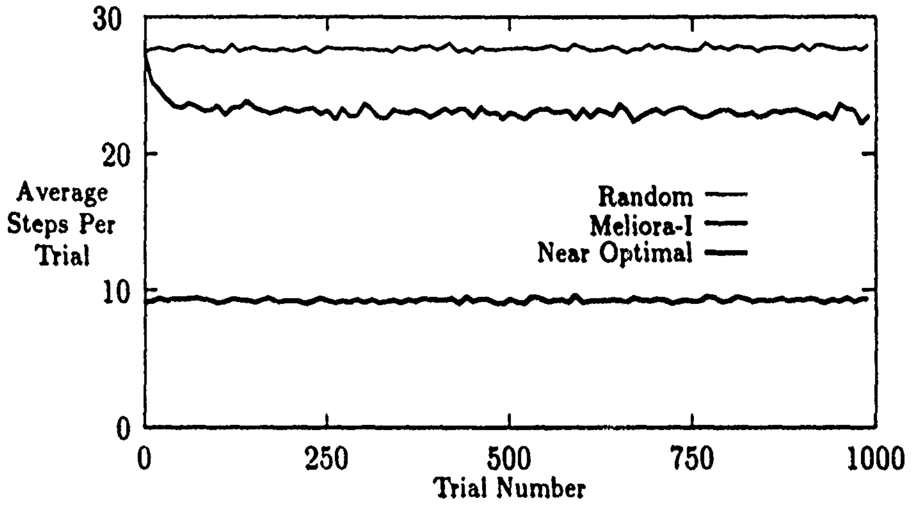


Figure 5.1: The number of steps taken per trial by Meliora-I versus trial number. Each point is averaged over 200 runs. Also shown are results for an optimal controller and a random controller. Meliora-I performs only slightly better than random and generally fails to reliably solve all but the easiest instances of the task.

Performance results for Meliora-I are shown in Figure 5.1. The figure shows the average, over 200 runs, of the number of steps taken per trial for a sequence of 1000 problem instances. These results are for $n_{quit} = 30$, $\alpha = 0.5$, $\gamma = 0.9$, and $p = 0.9$.[1] Also shown are plots for an agent acting randomly and an agent following the near-optimal policy shown in Figure 4.3. The plots show that 1-step Q-learning fails to learn the optimal policy (or a policy anywhere near it). Meliora-I's performance shows a slight initial improvement, but it quickly levels out at a performance that is only slightly better than random. By keeping track of the problem instances Meliora-I reliably learns to solve, we noticed that it manages only to learn the trivial problems in which the green block is initially clear. The learning of these instances accounts for the system's initial performance improvement. For all other instances, Meliora-I failed to learn a reasonable control strategy. In particular, it never reliably learned to uncover the green block.

[1]Other experiments showed the performance to be insensitive to variations in the parameter settings

5.2 Definitions and Nomenclature

Before we get into the details of why Meliora-I cannot solve the GB-task, it is useful to develop further our language for describing properties and relationships between internal and external decision problems.

To begin, it is convenient sometimes to use a single variable to denote a state-action pair. Thus, the term *decision* will be used as a synonym for "state-action pair," and the variable d will be used to denote the state action pair (s, a). Using this terminology, we say the action-value function is defined over the set of possible "decisions," $D = S \times A$.

Next, it is useful to define three sets that help to characterize the relationships between states and decisions in the external and internal decision problems. Given a formal description of an internal decision problem $(S_I, A_I, T_I\ R_I)$ stated in terms of an external model (S_E, A_E, T_E, R_E) and a sensory-motor model (P, I, M, C), define $SRep(s')$ to be the states in the external model that for one configuration or other of the sensory-motor system map into the internal state s'. That is, $SRep(s')$ denotes the class of external states represented by s'. Formally, $s \in SRep(s')$ if and only if there exists a sensory-motor configuration $c \in C$ such that $P(s, c) = s'$. If $s \in SRep(s')$ then we say that *s' represents s* and that *s is represented by s'*. Note that in general an internal state may represent many external states and an external state may be represented by many internal states, depending upon the configuration of the sensory-motor system.

Similarly, define $DRep(d')$ to be the decisions in the external model that, for one configuration or other of the sensory-motor system, map onto the internal decision $d' = (s', a')$. Formally, $d = (s, a) \in DRep(d')$ if and only if there exists a sensory-motor configuration $c \in C$ such that $P(s, c) = s'$ and $M(a', c) = a$. If $d \in DRep(d')$ then we say that *d' represents d* and *d is represented by d'*. Again in an active sensory-motor system, these relationships are generally many-to-many.

Finally, define $Cons(s, s')$ to be the sensory-motor configurations that map the external state s onto the internal state s'. Formally, $c \in Cons(s, s')$ if and only if $P(s, c) = s'$.

Given these definitions, it is now possible to introduce precisely the notion of *consistency*. In particular, an internal decision is said to be *consistent* if every decision it represents in the external model has the same optimal action-value. Formally,

$$d' \text{ is } \textit{consistent} \quad \text{iff} \quad \exists_{k \in \Re} \forall_{d \in DRep(d')} [Q^*_E(d) = k], \tag{5.1}$$

where $Q^*_E(d)$ is the optimal action-value for the external decision d. An internal decision that is not consistent is said to be *inconsistent*.

Similarly, an internal state is defined to be *consistent* if all of its corresponding decisions (state-action pairs) are consistent. Formally,

$$s' \text{ is } \textit{consistent} \quad \text{iff} \quad \forall_{a' \in A_I} d' = (s', a') \text{ is consistent.} \tag{5.2}$$

The concept of consistency allows us to sharpen our heretofore intuitive definitions of passive abstraction and perceptual aliasing. In particular, *passive abstraction* is the process of mapping, through perception, multiple external states onto a single *consistent internal state*. Conversely, *perceptual aliasing* occurs when multiple external states are mapped onto a single internal state in a way that results in an inconsistency.

5.3 The Effects of Perceptual Aliasing

Armed with these tools for describing various properties of internal decision problems, the poor performance of 1-step Q-learning on the GB-task can be explained, and it can be shown that most existing reinforcement learning algorithms cannot be used to learn to control agents with active sensory-motor systems.

The first observation to make about the internal decision problems of agents with active perception is that they are rife with perceptual aliasing and inconsistent internal states. Perceptual aliasing goes hand in hand with active perception since the whole point of active perception is to provide a flexible means for selectively sensing and ignoring selectively various aspects of the world. This includes the possibility of ignoring relevant information, which in turn leads to perceptual aliasing. Therefore, the internal decision problem facing the controller of any system that uses active perception will necessarily contain inconsistent internal states. Indeed, since in most cases careful control of the sensory system is required to register all of the relevant information, the *vast majority* of internal states are likely to be inconsistent.

Another observation to make is that internal decision problems that contain inconsistent internal states are necessarily non-Markov. This follows since, for inconsistent states, knowledge of the current state is not sufficient to characterize the dynamics of the process completely. Additional information, namely knowledge of the hidden, external state, can be used to improve predictions about the future of the process — a clear violation of the Markov property. This puts us on shaky ground with respect to 1-step Q-learning since it has only been shown to converge to the optimal policy for Markov decision problems. Indeed, as we shall see, inconsistent states and non-Markov decision problems wreak havoc on 1-step Q-learning.[2]

[2]When I say " it has only been shown to converge for Markov decision problems", I do not wish to minimize the significance of Watkins' convergence result for 1-step Q-learning. This

The trouble with inconsistent internal states is that they prevent the embedded decision system from accurately estimating and representing the *true* utility of applying an internal action at a given point in time. Given an inconsistent internal decision $d' = (s', a')$, the single action-value maintained by the Q-learner, denoted $Q_I(d')$, cannot simultaneously represent the disparate action-values for the external decisions in *DRep*(d'). This means that regardless of the value of $Q_I(d')$ there will be points in time when the internal state is s' and $Q_I(s', a')$ will not accurately reflect the true utility of executing the action a'. At these crucial instants in time, the internal action-value estimate and optimal action-value for the external decision it represents differ (or are inconsistent)! These events are called *utility aberrations.*

Utility aberrations are certain to occur in systems with inconsistent internal states and can impair decision making both locally and globally.

A local impairment occurs when, because of an inability to accurately represent the action-values of an inconsistent state, a non-optimal value is incorrectly assigned to the policy value of the inconsistent state. A local impairment can occur if either an optimal or non-optimal decision is inconsistent. If the optimal decision for a given state is inconsistent, and if the action-value estimate for that decision dips below the action-value estimate for a non-optimal action, because of a utility aberration, then the agent will incorrectly prefer the non-optimal action over the optimal one. Similarly, if a non-optimal decision is inconsistent and the utility aberration is such that its action-value estimate is increased over the action-value for the optimal decision, then the agent will incorrectly prefer the non-optimal action.

Utility aberrations can also have global effects, leading to non-optimal behavior in states that are otherwise consistent. Global impairments occur when inaccurate utility estimates from inconsistent states are used to update the action-value estimates for other (potentially consistent) states. For instance, in 1-step Q-learning, the action-value estimate for the state-action pair executed at time t is updated according to the rule,

$$Q_I(s_t, a_t) \leftarrow (1-\alpha)Q_I(s_t, a_t) + \alpha[r_t + \gamma U_I(s_{t+1})],$$

where $U_I(s_{t+1}) = \max_{a \in A_I} Q_I(s_{t+1}, a)$. Unfortunately, if s_{t+1} is inconsistent then $U_I(s_{t+1})$ may be inaccurate with respect to the current situation in the external world (due to a utility aberration) and the 1-step estimator, $r_t + \gamma U_I(s_{t+1})$, may be incorrect, thus introducing an error into $Q_I(s_t, a_t)$. This error may now lead

is an extremely general result and one of the few formal theorems in reinforcement learning. Indeed, Markov models are the foundation of most work on sequential optimization problems. Non-Markov models, because of their mathematical intractability, tend to be at the fringe. Unfortunately, active perception invariably leads to Non-Markov models. Therefore, we have no choice but to try to deal with them.

to non-optimal behavior in state s_t and an inaccurate estimate of $U_I(s_t)$, which in turn may infect other states, and so on.

5.4 A Simple Example

To illustrate the extent to which perceptual aliasing can interfere with Q-learning, let us consider a simple example. Consider the task shown in Figure 5.2. In this task, the external decision problem has a state space containing eight states, $S_E = \{s'_0, s'_1, s'_{2,5}, s'_3, s'_4, s'_6, g'\}$; two actions, $A_E = \{a_l, a_r\}$; and a deterministic transition function, shown in Figure 5.2a. The goal of the external task is to enter the goal state g, whereupon the agent receives a fixed reward $R_E(g) = 5000$. Non-goal states yield zero reward, $R_E(s_k) = 0$ for $k = 0$ to 6.

The optimal value function for the external task, denoted V_E^*, is an exponentially decreasing function of the distance to the goal. That is, $V_E^*(s) = R_E(g)\gamma^{(d(s)-1)}$, where $d(s)$ is the distance (in steps) from state s to the goal. The optimal policy, π_E^*, corresponds to choosing the action that minimizes the distance to the goal. In this case, the optimal policy requires the agent to moving right (a_r) at every opportunity (i.e., for all $s \in S_E$, $\pi_E^*(s) = a_r$). Notice that the optimal solution path for a given trial traces out a trajectory where $V_E^*(x_t)$ is monotonically increasing in time, and that the optimal policy corresponds to performing a gradient ascent of V_E^*. This result is illustrated in Figure 5.3a, which plots $V_E^*(x_t)$ versus time for a trial that begins in state s_0 at time $t = 0$ and follows the optimal trajectory to g at time $t = 7$. When applied directly to this problem, 1-step Q-learning can easily learn the optimal policy. However, let us introduce an inconsistency and see what happens.

Consider the internal decision problem that results when the agent's sensory-motor system implements a perceptual mapping that is fixed, one-to-one, and onto except for states s_2 and s_5, which get mapped onto the same internal state, $s'_{2,5}$. That is, let $S_I = \{s'_0, s'_1, s'_{2,5}, s'_3, s'_4, s'_6, g'\}$, where except for $s'_{2,5}$, s'_j (and g') represents world state s_j (and g). Also let the motor mapping be such that $A_I = \{a'_l, a'_r\}$, where a'_l and a'_r map to a_l and a_r, respectively. The transition diagram for this internal decision problem is shown in Figure 5.2b. Note that this problem is non Markov since the effects of actions are not independent of the past but depend upon the hidden, unperceived external state. Also note that a fixed optimal policy for this task is to always apply the action a'_r.

1-step Q-learning cannot learn the optimal policy for this task. In particular, if the agent's policy is initialized to the optimal policy and the controller is fixed so that the system follows the optimal policy with probability $p = 0.99$ and chooses a random action otherwise, and if the system is run for many trials and allowed to estimate the optimal value and action-value functions, then the following is observed. First, since the value and action-value estimates (U_I and Q_I respectively)

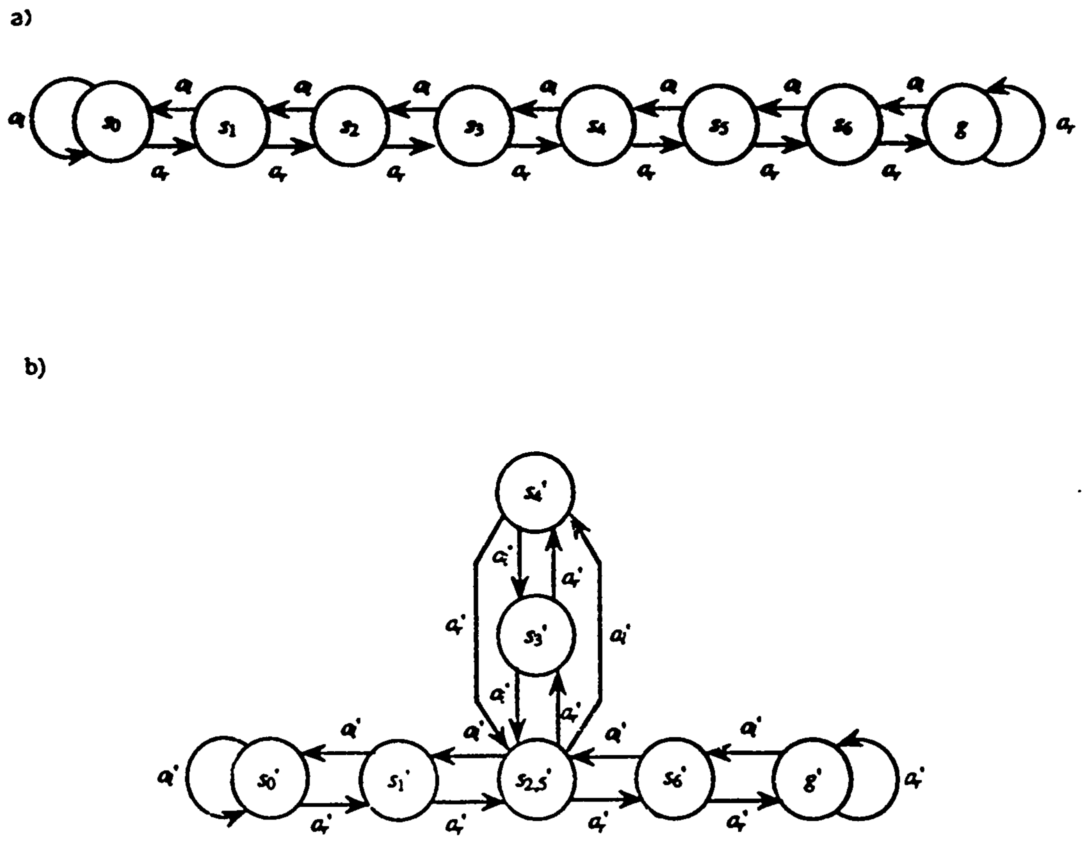
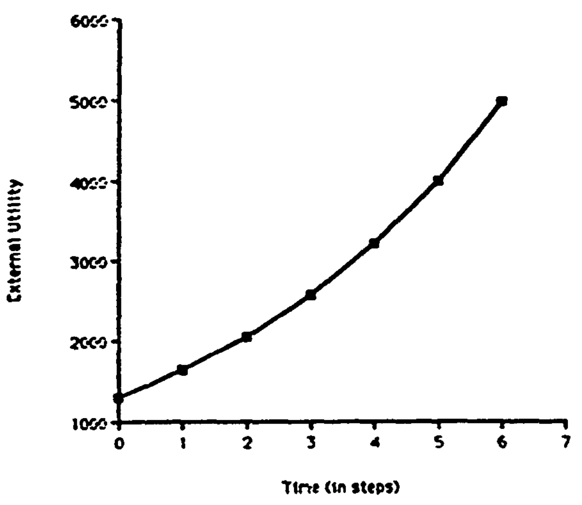


Figure 5.2: Transition diagrams for a simple decision task: a) the transition diagram for the external decision problem, b) the transition diagram for the internal (or perceived) decision problem when interpreted through a sensory-motor system with perceptual aliasing.

a)



b)

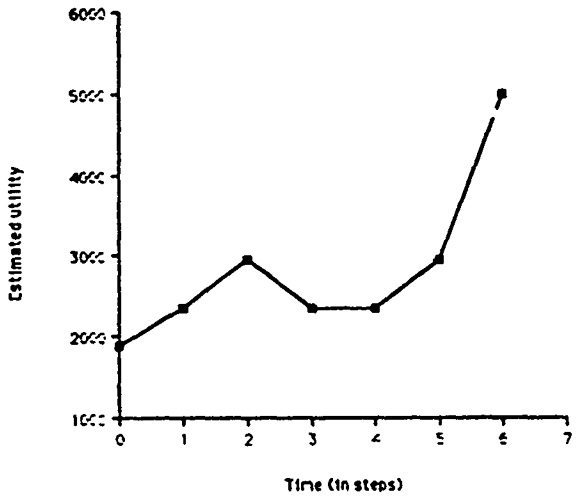


Figure 5.3: Plots of utility versus time as the agent traverses from state s_0 at $t = 0$ to g at $t = 7$ (for $\gamma = 0.8$): a) the utility for the external decision problem, V_E^*; b) the utility estimates for the internal decision problem, U_I, obtained by the 1-step Q-learning algorithm.

are based on *expected* returns, for the state $s'_{2,5}$, they take on values somewhere between the corresponding values for s_2 and s_5 in the external decision problem. That is,

$$V^*_E(s_2) \le U_I(s'_{2,5}) \le V^*_E(s_5), \tag{5.3}$$

$$Q^*_E(s_2, a_r) \le Q_I(s'_{2,5}, a'_r) \le Q^*_E(s_5, a_r), \tag{5.4}$$

and

$$Q^*_E(s_2, a_l) \le Q_I(s'_{2,5}, a'_l) \le Q^*_E(s_5, a_l). \tag{5.5}$$

Actually, the estimated action-value function does not even converge to the true sampled average of the returns observed. This follows since to update its action-value function, the agent uses a 1-step estimator which enforces only local constraints on the values estimated. If the learning rate is gradually decreased with time, the action-value function estimated by the agent converges to the values that satisfy the following local relationships:

$$\begin{aligned}
Q_I(s'_6, a'_r) &= R(g') + \gamma 0 = 5000 && (5.6)\\
Q_I(s'_{2,5}, a'_r) &= f_1[0 + \gamma U_I(s'_6)] + f_2[0 + \gamma U_I(s'_3)] && (5.7)\\
Q_I(s'_4, a'_r) &= 0 + \gamma U_I(s'_{2,5}) && (5.8)\\
Q_I(s'_3, a'_r) &= 0 + \gamma U_I(s'_4) && (5.9)\\
Q_I(s'_1, a'_r) &= 0 + \gamma U_I(s'_{2,5}) && (5.10)\\
Q_I(s_0, a'_r) &= 0 + \gamma U_I(s'_1) && (5.11)\\
Q_I(s'_6. a'_l) &= 0 + \gamma U_I(s'_{2,5}) && (5.12)\\
Q_I(s'_{2,5}, a'_l) &= f'_1[0 + \gamma U_I(s'_4)] + f'_2[0 + \gamma U_I(s'_1)] && (5.13)\\
Q_I(s'_4, a'_l) &= 0 + \gamma U_I(s'_3) && (5.14)\\
Q_I(s'_3, a'_l) &= 0 + \gamma U_I(s'_{2,5}) && (5.15)\\
Q_I(s'_1, a'_l) &= 0 + \gamma U_I(s'_0) && (5.16)\\
Q_I(s'_0, a'_l) &= 0 + \gamma U_I(s'_0) && (5.17)
\end{aligned}$$

where $U_I(x) = \max_{a \in \{a'_l, a'_r\}} Q_I(x, a)$.

In Equation 5.7, f_1 and f_2 are the fraction of times the application of a'_r in state $s'_{2,5}$ results in the next states being s'_3 and s'_6, respectively. Similarly, in Equation 5.13, f'_1 and f'_2 are the fraction of times the application of a'_l in state $s'_{2,5}$ results in states s'_1 and s'_4, respectively. If trials always begin in state s'_0, then $f_1 = f_2 = f'_1 = f'_2 = 50\%$ and the values for the utility and action-value functions will converge on the values shown in Table 5.1. Also shown in the table are the sampled utility and action-values (V_S and Q_S, respectively), obtained by actually measuring and averaging the returns received over many trials (instead of using a 1-step estimator). Notice that the sampled averages match the optimal values for

	s'_0	s'_1	s'_3	s'_4	$s'_{2,5}$	s'_6
$U_I(s)$	1882	2352	2352	2352	2941	5000
$Q_I(s,a'_l)$	1506	1506	2352	1882	1882	2352
$Q_I(s,a'_r)$	1882	2352	1882	2352	2941	5000
$V_S(s)$	1310	1638	2560	3200	3024	5000
$Q_S(s,a'_l)$	1048	1048	1310	1638	1935	3200
$Q_S(s,a'_r)$	1310	1638	2560	3200	3024	5000

Table 5.1: The utility and action-value functions estimated by the 1-step Q-learning algorithm and the true sampled utility and action-value functions. The estimated functions do not match the true sampled values since they are obtained by satisfying the local constraints imposed by the corrected 1-step estimator. The estimated utility and action-values are denoted U_I and Q_I, respectively, and the sampled utility and action-values are denoted V_S and Q_S. The values shown are for $\gamma = 0.8$.

the external decision task except for states s_2 and s_5, where utility aberrations occur. In this case,

$$V_S(s'_{2,5}) = 1/2V^*_E(s_2) + 1/2V^*_E(s_5) \tag{5.18}$$

$$Q_S(s'_{2,5},a'_l) = 1/2Q^*_E(s_2,a_l) + 1/2Q^*_E(s_5,a_l) \tag{5.19}$$

$$Q_S(s'_{2,5},a'_r) = 1/2Q^*_E(s_2,a_r) + 1/2Q^*_E(s_5,a_r). \tag{5.20}$$

Also notice that the utility and action-values estimated by 1-step Q-learning, except for state s_6, do not match either the external or the sampled utility and action values. This discrepancy arises because estimates for all the states up to s_5 (i.e., $s'_0, s'_1, s'_{2,5}, s'_3$, and s'_4) in the internal task are either directly or indirectly dependent upon the utility estimate for s_5. However, since s_2 and s_5 are indistinguishable, their internal action-value estimates are constrained to be the same. Thus, a utility aberration occurs whenever the world is in s_5. This inaccurate utility estimate in turn gets propagated back to all the states that precede it.

The next important observation to make is that the utility function (either learned or measured) for the internal decision problem is no longer monotonically increasing as the system traverses the optimal solution trajectory. This anomaly is shown graphically in Figure 5.3b, which plots $U_I(x_t)$ as a function of time as the system follows the optimal trajectory from s'_0 to g'. The plot shows that a utility aberration occurs at $t = 2$ when the system first encounters $s'_{2,5}$. In reality, the world is in state s_2 and the true expected return is $V^*_E(s_2) = 2048$ (for $\gamma = 0.8$). However, because s_2 and s_5 are indistinguishable in the internal representation, the internal decision system overestimates the expected return at $t = 2$. Similarly, another utility aberration occurs the second time $s'_{2,5}$ is encountered, at $t = 5$ when the world is in state s_5. In this case, $U_I(s'_{2,5})$ underestimates the expected return.

If we relax our hold on the decision policy and allow the system to adapt, we find that the optimal policy is unstable! Not only is the system unable to find the optimal policy, it actually moves away from it. In general, the system will oscillate among policies, never finding a stable one. The instability can be understood by considering the effect of utility aberrations on the policy. Recall that in Q-learning the system locally adjusts its policy in order to maximize the expected return. Thus, after running the agent with a fixed policy for many trials and then releasing it, the policy value for state s'_3 will be changed so that the system tends to take actions that move it back to $s'_{2,5}$ instead of forward to s'_4 (since $Q_I(s'_3, a'_l) > Q_I(s'_3, a'_r)$). The large utility value for state $s'_{2,5}$ acts as an attractor for nearby states, such as s'_3, and causes them to change their local policy away from optimal. An intuitive way to understand the problem is to consider a local homunculus that sits at s'_3 and can see the utilities of its neighbors. From his point of view, $s'_{2,5}$ looks more desirable than s'_4 since once the system is in $s'_{2,5}$ it can execute a'_r, which often leads to s'_6 (one step from the goal). On the other hand, choosing the action which leads to s'_4 leaves the system still three steps from the goal. From the homunculus' point of view, going to $s'_{2,5}$ is on average better than going to s'_4. What the homunculus cannot perceive (because of perceptual aliasing) is that going from s'_3 directly to $s'_{2,5}$ always returns the real external world to state s_2, which cannot reach s_6 directly. The problem is that the homunculus cannot distinguish between s_2 and s_5, as they are both represented by $s'_{2,5}$, and it erroneously makes the Markov assumption — that the effects of actions depend only upon the current perceived state.

The utility aberrations are also unstable since they are based on a running average of the expected returns. If, because of policy changes, s_5 is rarely visited, the aberration at s_2 will disappear. Unfortunately, as soon as the policy is changed so that s_5 begins to be encountered more frequently, the aberration reappears, and so on. Thus, the system oscillates from policy to policy, unable to converge on a stable one.

5.5 Looking Back at Meliora-I

It is now easy to see why Meliora-I cannot learn to solve the GB-task. In Meliora-I, the internal state space is rife with inconsistencies. Indeed, the only time the internal state is consistent is when one of the markers (either the action or the attention-frame) is on the green block. All other configurations of the sensory-motor system generate inconsistent internal states.

An example of an inconsistent internal state is shown in Figure 5.4. This figure shows four external states, each with different optimal utilities, that under appro-

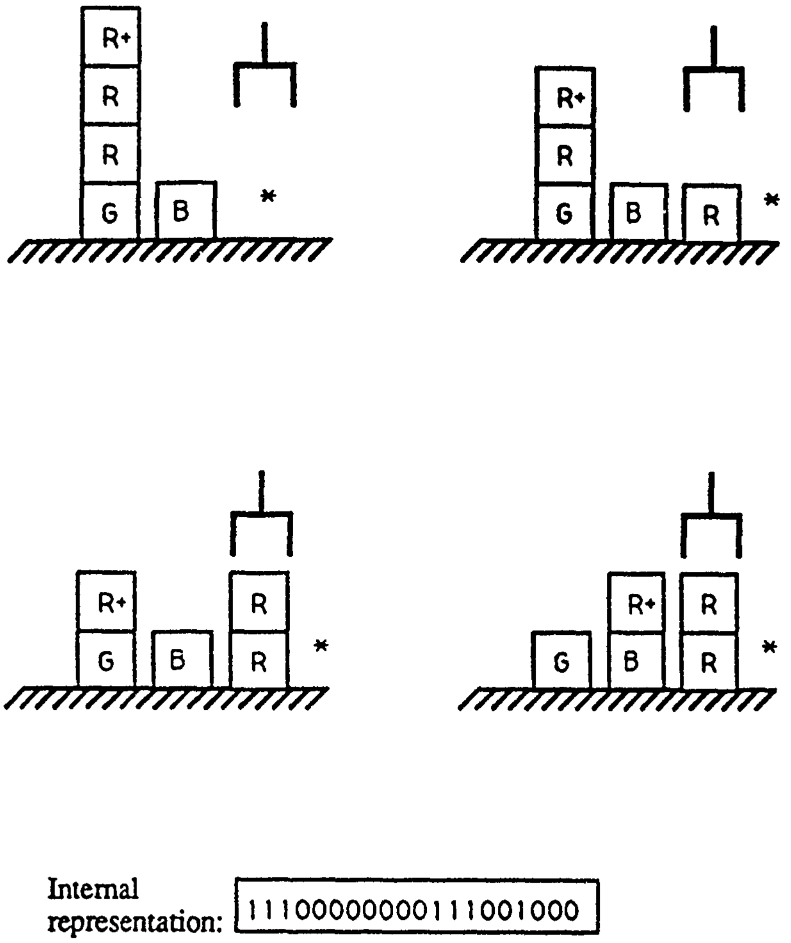


Figure 5.4: Four different external states, each with different utilities, can generate the same internal state. In this case, the distance to the goal state for each of these states is 15, 12, 7, and 2 steps, respectively. The utility estimate for the internal state tends to reflect an average of the utilities of the external states it represents. If we just consider these four external states (indeed the equivalence class for this internal state contains many more external states), then the utility of the internal state would correspond roughly to a state approximately 9 steps away from the goal. When representing external states like those on the top, the internal state tends to overestimate the utility of the situation (aberrational maxima) and when representing situations like those on the bottom, the internal state tends to underestimate the utility of the situation (aberrational minimum).

priate conditions generate the same internal state (shown below). The utility and action-values for this internal state tend to reflect an average of the utilities and action-values for the external states it represents. When encountered in operation, it is unlikely that this state's utility estimate will match the actual utility of the current external state — sometimes it will overestimate and sometimes it will underestimate.

To see how this inconsistent state interferes with decision making, consider the situation shown at the top in Figure 5.5. In this case, the green block is covered by three blocks, the action-frame marks the top block, and the attention-frame marks the green block. The internal state generated by this arrangement is consistent and shown directly below the pile. Consider the result of performing two different actions. First, suppose the agent performs the grasp action. This action results in the external and internal states shown on the left. The internal state for this situation is also consistent. Now suppose that instead of performing the grasp action the agent performs the *move-attention-frame-to-table* action. This action results in the external and internal states shown on the right Figure 5.5. In this case, the internal state is the inconsistent state from Figure 5.4. Even though the utility of the external state on the left in Figure 5.5 is greater than the external state on the right (a minimum distance of 16 versus 14 steps), Meliora-I prefers *move-attention-frame-to-green* over *grasp-object-at-action-frame*. This follows since the utility estimate for the internal state on the right, due to averaging, tends to be higher than the utility of the consistent internal state on the left. The situation on the left corresponds to a state of knowledge in which the agent's internal state encodes the truth about the utility of the world, whereas the situation on the right corresponds a state of ignorance. When viewed by an outside observer, Meliora-I appears to take an "ignorance is bliss" approach to control. That is, when it encounters a situation in the external world where it discovers the utility is low (i.e., a great deal of work must be done to obtain some reward), instead of gritting its teeth and getting on with it, it prefers to ignore the problem by focusing its attention elsewhere (i.e., since states of ignorance appear to be better off on average than states that are known to be far from the goal).

5.6 The Long Arm of Perceptual Aliasing

The difficulties caused by perceptual aliasing and inconsistent internal states are not unique only to 1-step Q-learning. Indeed, any reinforcement learning algorithm that uses any form of truncated corrected return is subject to the detrimental effects of perceptual aliasing. This includes the whole family of Q-learning algorithms, algorithms based on temporal difference methods, and the bucket brigade algorithm commonly used in classifier systems.

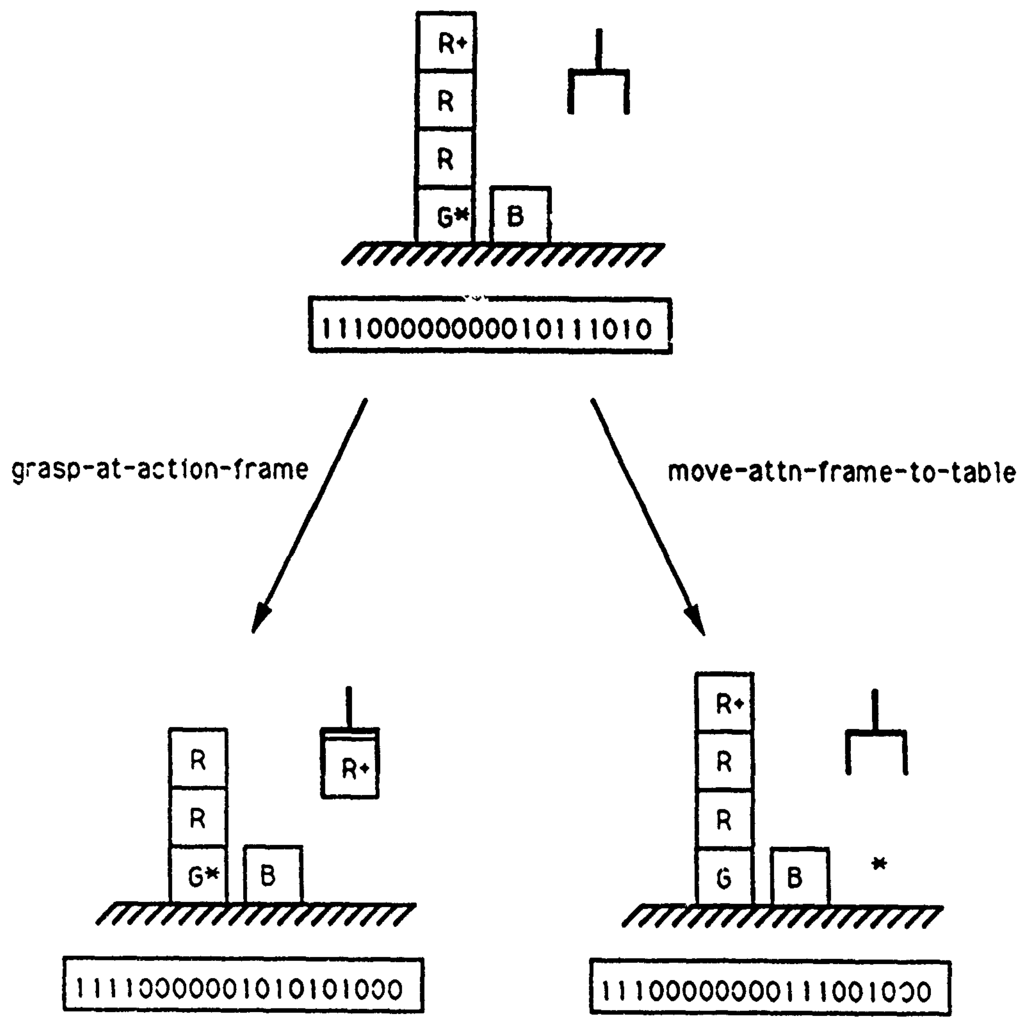


Figure 5.5: Three pairs of external and internal states from the GB-task. External states are depicted as piles, and internal states are depicted as bit vectors. The + indicates the position of the action-frame; the * indicates the attention-frame. At the top is a situation in which the attention-frame is bound to the green block and the action-frame is bound to the top block in the stack; the corresponding internal state is consistent. On the left is the situation that results from executing the optimal action, *grasp-at-action-frame* in the top state; this internal state is also consistent. On the right is the situation that results from executing the *non-optimal* action *move-attention-frame-to-table*; this internal state is inconsistent, and, due to utility aberrations, tends to have a higher utility estimate than the internal state on the left.

Also, as previously mentioned, perceptual aliasing always accompanies adaptive systems that use active perception since in general a flexible sensory-motor system can always be configured to ignore relevant information. Indeed, inconsistent states are likely to be pervasive since in many cases only careful configuration of the sensory-motor system will lead to consistent internal states. For instance, in the GB-task, consistent internal states are achieved only when the green block is marked. All other configurations lead to inconsistencies.

Finally, the negative effects of perceptual aliasing need not arise only in systems with active perception. In general, perceptual aliasing accompanies all abstraction or generalization mechanisms — that is, any time it is possible to ignore information that is relevant to decision making and utility estimation. For instance in an example similar to the one described in Figure 5.2, Grefenstette [Grefenstette, 1988] has shown how strength averaging in the rules of a classifier system prevents the system from learning an optimal control strategy. In this case, rules that match multiple world states (allowed to improve generalization) exhibit perceptual aliasing and, as a result, are vulnerable to inconsistencies and inaccurate utility estimates. Other, similar examples of perceptual aliasing have been described recently by Chapman and Kaelbling [Chapman and Kaelbling, 1991] and Tan [Tan, 1991a; Tan, 1991b].

6 Learning Consistent Representations

This chapter describes Meliora-II, a program that successfully learns the GB-task [Whitehead and Ballard, 1990; Whitehead and Ballard, 1991a]. The basic idea underlying Meliora-II is to learn and use an internal representation that is complete and consistent. Instead of freely mixing perceptual and overt actions in a monolithic controller that permits inconsistent states in the internal representation, Meliora-II divides control into two distinct phases: state identification and overt control. During state identification, a consistent internal state is generated by executing perceptual actions in order to configure the sensory-motor system properly. This consistent internal state is then made available to an overt controller, which generates the next overt action. Both the state identification and overt control processes are adaptive. The state identification module is adapted when it is found to generate internal states that are inconsistent. The overt control module is adapted based on rewards gained through interactions with the world.

In Meliora-II overt control is achieved by using a slightly modified version of 1-step Q-learning, which aims to learn the internal equivalent to the external optimal policy. State identification is achieved by learning a perceptual policy that maps input vectors into perceptual actions. To generate an internal state, a sequence of perceptual actions is performed and a set of candidate internal states (state vectors) is generated. One of these internal states is then chosen to represent the current external world state. The detection of inconsistent internal states, used to update the perceptual policy, is accomplished by monitoring the sign in the estimation error in the 1-step Q-learning rule. These specific techniques are adequate for Meliora-II and the GB-task; however, other more general techniques exist as well. In general, Meliora-II represents a specific example of a general approach to the adaptive control of perception and action that we call the *Consistent Representation* (or CR) Method. Algorithms and architectures based on this method all share the same basic idea, which is to break control into state identification and overt control and to generate an internal representation that is consistent at each point in time. After describing the specific algorithm used by

Meliora-II and some experimental results, the discussion turns to the CR-method in general. This discussion includes a description of the basic architecture of systems that use the CR-method; a review of Meliora-II in terms of this architecture; a discussion of some alternatives to the algorithms used in Meliora-II; and a brief review of two other systems that also employ the CR-method. The chapter concludes with a discussion of the limitations of Meliora-II and of the CR-method in general.

6.1 Meliora-II

Meliora-II represents our solution to the problem of perceptual aliasing. That is, Meliora-II's decision system is designed specifically to be embedded within an agent with an active sensory-motor system and to control perception actively to overcome the negative effects of perceptual aliasing.

The decision system used in Meliora-II is based on three tenets:

1. In active perception a world state can be represented by multiple internal states, one of which is usually consistent. That is, if the agent looks around enough it will eventually attend to those objects that are relevant to the task, and the internal state associated with that sensory configuration will be consistent. Our algorithm depends on the existence of one consistent internal state for each world state.

2. Inconsistent states disrupt the decision system's ability to learn by promising inaccurate estimates of the expected return. Detecting inconsistent states and eliminating their participation in decision making and action-value estimation minimizes their negative effects.

3. If the world is deterministic, then inconsistent states will (because of averaging) periodically overestimate the utility of the actual world state, whereas the incidence of overestimation in consistent states can be made to diminish with time. Therefore, inconsistent states can be detected by monitoring the sign of the estimation error in the 1-step Q-learning rule, Equation 3.40.

6.1.1 The Overt Cycle

The decision procedure used by Meliora-II is called the *Lion algorithm* and is outlined in Figure 6.1. The main loop in the decision procedure is the overt cycle, which concerns itself with choosing overt actions in an attempt to maximize the expected future return. Embedded within the overt cycle is a perceptual cycle (the identification stage). After each overt action, the system executes a sequence

```
Overt Cycle:
  1) Execute Perceptual Cycle and generate $S_t$, a set of internal
     representations for the current world state.
  2) Choose an internal state, the lion, to represent the current world state by selecting
     the state with the largest overt utility estimate:
     $lion = \arg\max_{s \in S_t}[V_I(s)]$.
  3) Estimate the utility of the current world state, $s_t$: $V_E(s_t) \leftarrow V_I(lion)$.
  4) Execute Update-Overt-Q-Estimates based on $V_E(s_t)$, $r_{t-1}$, $oact_{t-1}$ and $lion_{t-1}$;
     where $r_t$ is the reward received at time $t$, $oact_{t-1}$ is the last overt action executed,
     and $lion_{t-1}$ is the internal state selected to represent the previous world state.
  5) Choose the next overt action to execute:
     With probability $p$ follow policy $f_o(lion)$,
       $oact \leftarrow \arg\max_{a \in A_O}[Q_I(lion, a)]$.
     Otherwise choose randomly: $oact \leftarrow Random(A_O)$
  6) Execute $oact$ to obtain a new world state and a reward $r_t$.
  7) Go to 1).

Update-Overt-Q-Estimates:
  1) Estimate the error in the lion's action-value: $E_{lion} \leftarrow (r_{t-1} + \gamma V_E(s_t)) - Q_I(lion_{t-1}, oact_{t-1})$.
  2) Update the action-value of the lion:
     If ($E_{lion} < 0$) then the lion is suspected of being inconsistent, so suppress it:
       $Q_I(lion_{t-1}, oact_{t-1}) \leftarrow 0.0$
     Else update it using the standard 1-step Q-learning rule:
       $O_I(lion_{t-1}, oact_{t-1}) \leftarrow Q_I(lion_{t-1}, oact_{t-1}) + \alpha E_{lion}$.
  3) Update non-lion internal states:
     For each $s \in S_{t-1}$ and $s \neq lion_{t-1}$ do:
       Let $E_s = r_{t-1} + \gamma V_E(s_t) - Q_I(s, oact_{t-1})$
       If ($E_s > 0$) then $s$ is suspected of being inconsistent, so suppress it:
         $Q_I(s, oact_{t-1}) \leftarrow 0.0$
       Else update it using the lion's error:
         $Q_I(s, oact_{t-1}) \leftarrow Q_I(s, oact_{t-1}) + \alpha' E_{lion}$ — where $\alpha' < \alpha$.

Perceptual Cycle:
  1) Initialize $S_t$: $S_t \leftarrow \{s_c\}$, where $s_c$ is the current internal state.
  2) Do $n$ times: (in our implementation $n = 4$)
     i) Select the next perceptual action:
       With probability $p'$ follow the perceptual policy: $pact \leftarrow f_p(s_c)$,
       where $s_c$ is the current input vector and $f_p(s_c) = \arg\max_{a \in A_P}[Q_I(s_c, a)]$.
       Otherwise choose randomly: $pact \leftarrow Random(A_P)$.
     ii) Execute $pact$ to obtain a new internal state $s'$.
     iii) Update the action-value estimate for the perceptual decision $(s_c, pact)$:
       $Q_I(s_c, pact) \leftarrow Q_I(s_c, pact) + \alpha(V_I(s') - Q_I(s_c, pact))$.
     iv) Add $s'$ to $S_t$: $S_t = S_t \cup \{s'\}$
     v) Update $s_c$: $s_c \leftarrow s'$.
  3) Return $S_t$
```

Figure 6.1: An outline of the decision procedure implemented by Meliora-II. This procedure is called the *lion algorithm* and is designed specifically to overcome the difficulties caused by perceptual aliasing.

of perceptual actions (the perceptual cycle) in an attempt to identify an internal state that consistently represents the current external state. The input vectors observed during the t^{th} perceptual cycle define a set of candidate internal states, S_t. Each of these internal states corresponds to a different view (representation) of the current external world. The state chosen to represent the current situation is called the *lion*[1] and is simply the internal state in S_t with the largest utility estimate. That is,

$$lion = \arg\max_{s \in S_t}[V_I(s)] \tag{6.1}$$

where $V_I(s)$, the overt utility estimate, is the maximum overt action-value for state s. That is,

$$V_I(s) = \max_{a \in A_O}[Q_I(s,a)]. \tag{6.2}$$

Once the lion is selected, the utility of the current external state, $V_E^*(x_t)$, is estimated using the overt utility estimate for the lion, $V_I(lion)$. As will be described below, the algorithm for adjusting the action-value estimates for overt actions severely lowers the values for inconsistent decisions. Consequently, the lion tends to be consistent and overt control tends to be based on action-value estimates that accurately reflect the true state of the external world.

Once $V_E^*(x_t)$ has been estimated, action-value estimates for the previous overt action are updated (as described below). The overt cycle then continues by selecting an overt action to execute. With probability p, the system chooses the action consistent with its overt policy f_o; the rest of the time it chooses an overt action at random. When following policy, the action executed is simply the overt action that maximizes the action-value function for the lion:

$$oact = \arg\max_{a \in A_O}[Q_I(lion,a)]. \tag{6.3}$$

Once an overt action is chosen, it is performed and the overt cycle begins anew. Figure 6.2 shows a cartoon of the decision system in action. The large nodes represent external world states, and the arcs between them overt actions. Embedded within each large node is a subgraph representing the perceptual cycle. The nodes in this graph correspond to internal states seen by the embedded decision system, and the arcs between them correspond to perceptual actions.

6.1.2 Learning a New Action-Value Function

In Dynamic Programming and Q-learning the action-value of a decision, for a given policy, is defined as the return the system expects to receive given that it performs that decision and follows the policy thereafter (*cf.* Equation 3.13). However, for inconsistent internal decisions this definition leads to inaccurate action-values (utility aberrations). Meliora-II uses a modified learning algorithm that

[1]Since it takes the lion's share of credit or blame for the agent's performance.

is based on Q-learning but incorporates a competitive component. This component tends to suppress the action-values of inconsistent decisions while allowing action-values for consistent decisions to take on their nominal values. This allows consistent states to be selected during state identification and provides the overt cycle with action-values that accurately reflect the expected returns of the underlying external decision problem.

The algorithm for updating Q_I is based on the following ideas: 1) the lion state chosen to represent the current world state should be consistent and its action-values should take on their nominal values; 2) the action-values for other internal states in S_t that might otherwise be used to represent the current state do not need to have their action-values updated as long as the action-values for the lion are accurate; 3) any time a decision is inconsistent it should be detected and its action-value should be suppressed. Ideally, the decision system should learn a new action-value function in which the action-values of consistent decisions take on their corresponding values for the external task and the action-values for inconsistent decisions are zero:

$$Q_I^{ideal}(s,a) = \begin{cases} Q_E^*(s_E, a_E) & \text{if } (s,a) \text{ is consistent} \\ 0 & \text{otherwise} \end{cases} \tag{6.4}$$

where $(s_E, a_E) \in DRep(s,a)$. Since rewards are always positive in the GB-task, Q_E^* is uniformly greater than zero. Thus, when using Q_I^{ideal} to define its overt policy Meliora-II would never base its decision on an inconsistent internal state and would always choose the optimal overt action.[2]

In Meliora-II, inconsistent decisions are detected as follows. If at time t the action-value for any decision $d = (s,a)$ where $s \in S_t$, is greater than the estimated return obtained after one step, $r_t + \gamma V_I(lion_{t+1})$, then the decision is suspected of being inconsistent and its action-value is suppressed (e.g., reset to zero). Actively reducing the action-values of lions that are suspected of being inconsistent gives other (possibly consistent) internal states an opportunity to become lions. If the lion does not overestimate, its action-value is updated using the 1-step Q-learning rule. To prevent inconsistent states from climbing back into contention and competing for lionhood, the estimates for non-lion decisions in S_t are updated at a lower learning rate and only in proportion to the error in the lion's action-value. Also, any time a decision's action-value overestimates the 1-step return, it is suppressed. This algorithm works for external tasks that are deterministic and have only positive rewards (e.g., the GB-task). For these cases, the property that allows the algorithm to work is that inconsistent decisions will eventually overestimate their action-values (due to utility aberrations). Thus, inconsistent states will eventually be suppressed. On the other hand, it can be shown that a

[2]Notice that this definition of optimal does not account for the cost of perceptual actions. Here optimality is defined in terms of the external decision problem.

consistent lion is stable if every external state between the lion and the goal (or every state in the limit cycle of the optimal policy) also has consistent lions with accurate action-value estimates.[3] Thus, inconsistent decisions are unstable with respect to lionhood while consistent decisions eventually become stable. The steps for updating action-values are shown in Figure 6.1 under the **Update-Overt-Q-Estimates** heading.

6.1.3 The Perceptual Subcycle

The steps in the perceptual cycle are sketched in Figure 6.1 under the **Perceptual Cycle** heading. The objective of the perceptual cycle is to accumulate a set of internal representations of the external world, one of which is consistent. This goal is achieved by executing a series of perceptual actions. In Meliora-II the perceptual cycle executes a fixed number ($n = 4$) of perceptual actions. This number has proven adequate for the GB-task, but it is easy to imagine variable length perceptual cycles in which the cycle either terminates as soon as a consistent internal state is found or increases when inconsistent states are encountered. The algorithm for selecting actions within the perceptual cycle is similar to the algorithm for choosing overt actions in the overt cycle. With probability p' (e.g., $p' = 0.9$), the system follows its perceptual policy f_p; otherwise, it selects a perceptual action at random. When following policy, the action selected is the perceptual action with the maximal action-value for the currently perceived input vector:

$$f_p(s_p) = \arg\max_{a \in A_P} [Q_I(s_p, a)] \tag{6.5}$$

where s_p is the currently perceived input vector.

The rules for updating action-values for perceptual actions are shown in Figure 6.1 within the **Perceptual Cycle** procedure. These updating rules lead to action-values that *average* the overt utilities of the internal states that result from executing a perceptual action. Since consistent states tend to have higher overt utilities than inconsistent states (whose action-values are suppressed), the effect is to choose perceptual actions that lead to consistent internal states.

6.2 Performance Results

A series of experiments were performed on the GB-task to evaluate its performance quantitatively. In each run, the robot was sequentially presented with 1000 instances of the task (i.e., 1000 trials). As in Chapter 5, each instance consists of a randomly configured pile of 4 blocks, with the pile always containing exactly one

[3]See [Tan, 1991a] for a nice analysis of this overestimation technique.

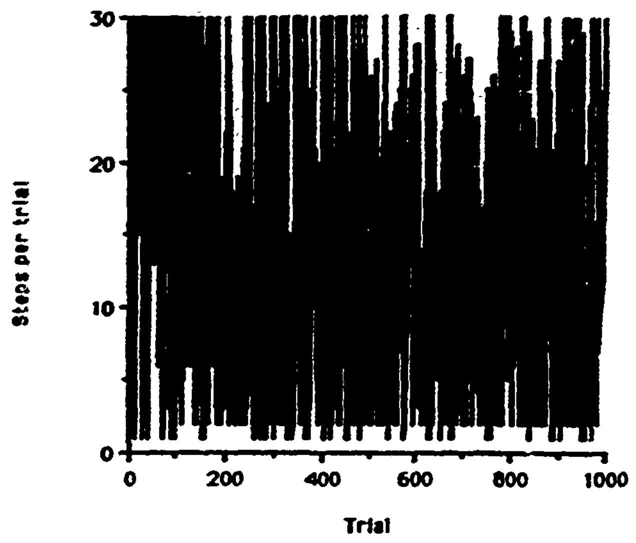


Figure 6.3: A plot of the number of steps per trial as a function of the instances seen by Meliora-II for a typical experimental run. High variation is due to the wide variety of tasks being solved.

green block; and if in any trial the robot fails to solve the problem after $n_{quit} = 30$ overt actions, it moves on to the next trial.

Performance results for a typical experimental run are shown in Figure 6.3. The graph shows the number of overt actions taken by Meliora-II for each of the 1000 instances of the task it encounters during a typical run. Initially, Meliora-II fails on almost every trial (i.e., it takes 30 steps and quits). It does, however, manage to solve a few instances. These early successes are invariably easy problems, requiring only one or two correct actions to solve. After about 100 trials, Meliora begins to solve more and more instances including more difficult problems. Eventually, it learns to solve even the most difficult instances and rarely fails (e.g., $< 5\%$ failure after 1000 trials).

Meliora's performance on a given trial depends strongly on the difficulty of the trial instance; consequently, the curve in Figure 6.3 has a large variance. A clearer picture of Meliora's performance is obtained by averaging results over multiple experimental runs. Figure 6.4 plots the solution time per trial averaged over 200 runs. Plots for the optimal number of steps (average of 200 runs) and for an agent behaving randomly are also shown. The figure clearly shows that Meliora's initial performance is poor — near the maximum of 30 steps per trial — but improves considerably during the first few hundred trials. The system's performance settles at just under 12 steps per trial (about 125% optimal).

The system's performance fails to converge to optimal for two reasons. First, with probability $1 - p$ ($p = 0.9$), the decision system chooses its overt action ran-

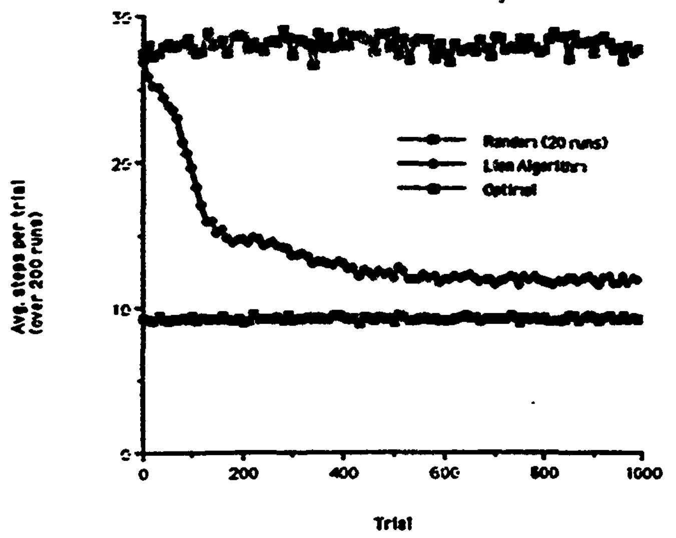


Figure 6.4: A plot of the average number of steps per trial as a function of the instances seen by Meliora-II. The average is taken over 200 runs and provides a smoother picture of the system's learning curve. The system's steady state performance is approximately 125% optimal. Also shown is the average number of steps taken by an agent acting randomly.

domly, reflecting a simplification in our decision algorithm that can be eliminated by incorporating more complex procedures for controlling exploration [Barto *et al.*, 1990; Kaelbling. 1990]. Second, the decision system is not guaranteed in every case to find a consistent lion (even if it exists) since the perceptual subcycle only executes 4 perceptual actions and chooses the lion from the set of at most five unique internal states. Further, perceptual actions are also occasionally $(1 - p' = 0.1)$ selected randomly. As a result, residual inconsistent lions occasionally arise and interfere with the system's performance.

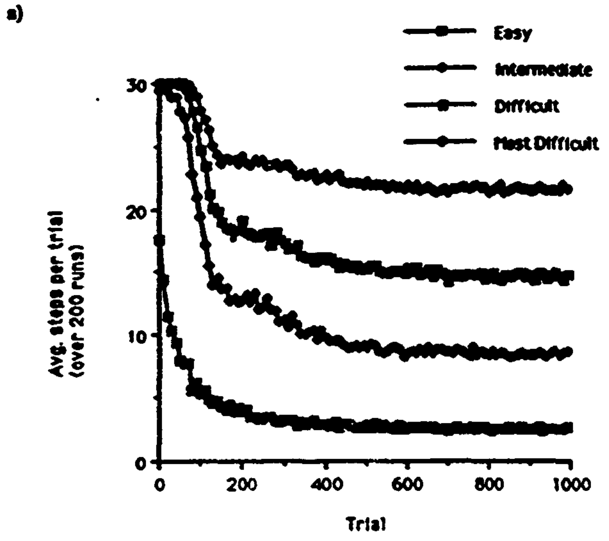
Figures 6.3 and 6.4 show that the system learns to solve the task, but they say nothing about which instances Meliora learns to solve first or the order in which the robot learns its task-dependent representation. To get a glimpse at the order in which instances of the task are learned, each problem instance was classified into one of four categories: easy, intermediate, difficult, and very difficult. Easy problems correspond to instances in which the green block is clear and the robot need only pick it up. Intermediate problems include instances where the green block is covered by one block; difficult problems, two blocks; and very difficult problems, three blocks. Plots of the average trial times and average success rate for each of these four classes of problems are shown in Figure 6.5 and Figure 6.6, respectively. Both figures show that the agent first learns to solve easy tasks reliably, and then learns more and more difficult ones. In Figure 6.5, the agent shows improvement on easy tasks immediately; it shows improvement on intermediate tasks after 10-20 trials; on difficult tasks after 50-60 trials; and on the most difficult tasks after 70-80 trials (see Figure 6.5b). A similar trend is seen in Figure 6.6, which also shows that the agent eventually learns to solve all but the most difficult tasks reliably and then only fails about 10% of the time.[4]

To determine the order in which Meliora-II learns a consistent representation, statistics were collected to measure the amount of overestimation that occurs during learning. As before, world states were classified into four categories according to their distance to the goal: easy, intermediate, difficult, and most difficult. For each class, the fraction of times per trial (over 200 runs) the lion overestimated (and was suppressed) was maintained as a function of the number of trials seen. These percentages are plotted in Figure 6.7. As expected, the agent initially overestimates a high fraction of the time. This fraction is especially high because a single overestimation can cause a chain of subsequent overestimations; and lacking knowledge on how to control perception, the agent frequently fails to choose a consistent lion. With experience, however, the agent eventually learns to select consistent internal states, and the amount of overestimation decreases.

We expected Meliora-II to learn consistent lions for easy states first and then to boot-strap its way to consistency for more and more distal states. To some extent this expectation is verified in Figure 6.7, which shows that the amount of overestimation decreases first for easy states and decreases later for more difficult states. Early on, the fractions for intermediate, difficult, and most difficult problems are virtually indistinguishable, explained by the fact that initially these more difficult problems are rarely solved, and when they are, they tend to be inefficient. For example, when solving an intermediate problem, it is common for the agent to stack an extra block on the green pile, try other unhelpful actions within that configuration for a while, unstack the block, and go on to solve the problem. Thus, the agent sees mixes of intermediate, difficult, and most difficult states. Initially, therefore, all trials end up visiting about the same fraction of consistent states. This random searching is much less prevalent in easy tasks whose solutions involve only one or two correct actions. Eventually, as the agent learns to solve easy problems (after 80-100 trials), intermediate states become increasingly consistent and the agent visits harder states less frequently on its way to the goal. The inconsistency in intermediate states tends to decrease while the consistency of more difficult states remains unchanged.

Figure 6.7 also shows that after 1000 trials the agent continues to overestimate a substantial fraction of the time. This fraction is fairly low for easy problems ($\approx$ 5%) but higher for the most difficult problems ($\approx$ 45%). There are three reasons for this high rate of overestimation. First, as previously mentioned, Meliora-II is

[4]Increasing n_{quit} slightly, say to 40, almost always gives the agent the extra time it needs to solve even the most difficult problems



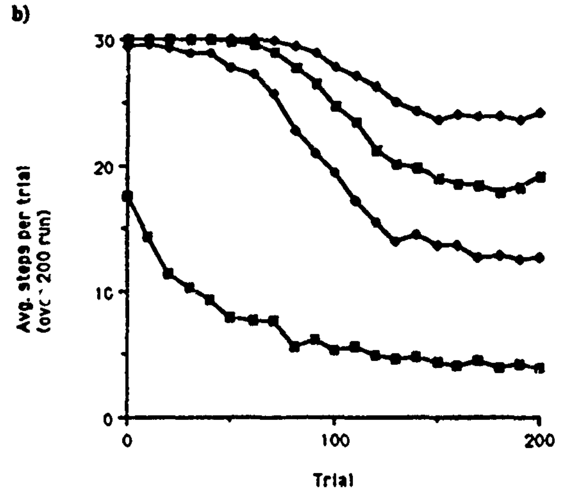
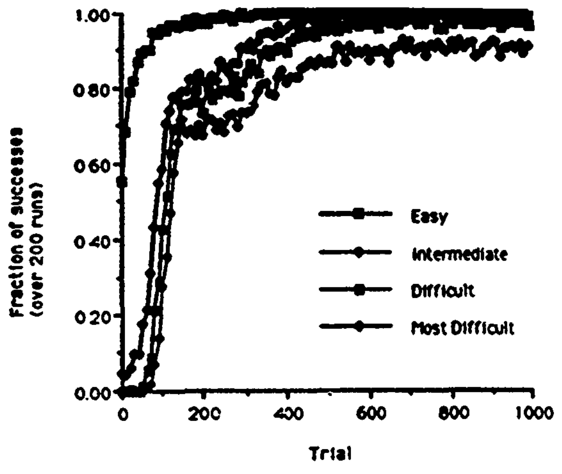


Figure 6.5: Plots of the average number of steps per trial for each of the four classes of problem instances: i) easy (no unstacking); ii) intermediate (1 to unstack); iii) difficult (2 to unstack); and iv) most difficult (3 to unstack). a) shows a complete plot ranging from 0 to 1000 trials; b) shows a focused plot ranging from 0 to 200 trials. The plots show that the agent learns to solve easier tasks first.

a)



b)

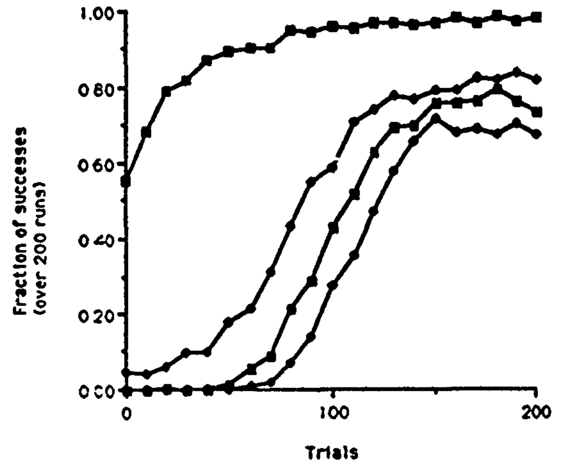


Figure 6.6: Success rates for each of the four classes of problem instances versus the number of trials seen by the agent. a) shows a complete plot ranging from 0 to 1000 trials; b) shows a focused plot ranging from 0 to 200 trials. The plots show that the agent learns to solve easier tasks first and eventually learns to solve all instances fairly reliably.

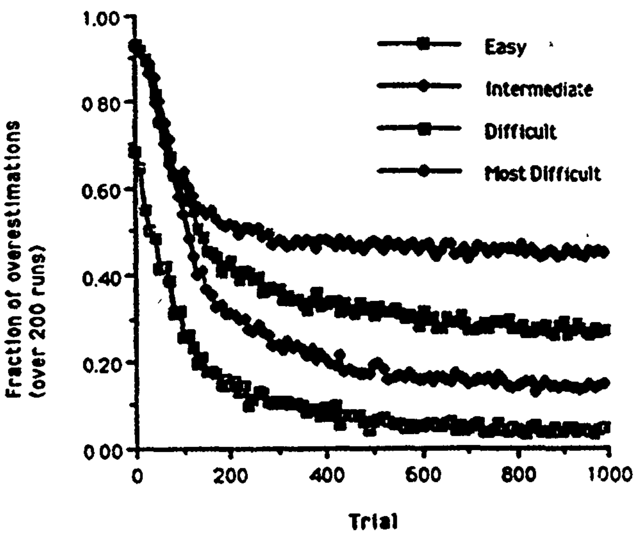


Figure 6.7: The fraction of overestimations encountered over 200 runs for each of the four classes of problem instances. The plot shows that consistent representations are learned for easy problems first, followed by consistent representations for more difficult problems, and that the agent continues to perform in the face of residual inconsistencies and overestimation.

not guaranteed always to find a consistent internal state, even if one exists. This explains the small fraction of steady state overestimation that occurs even for easy problems. Second, a single overestimation (and suppression) tends to cause a chain reaction of overestimations in earlier "set-up" states (even for consistent states). Thus, the high fraction of overestimation in more distal (difficult) states is explained by the fact that occasional overestimations in easy states propagate back to these states and destroy consistencies there. Third, when overestimations occur they tend to impair the agent's decision policy temporarily. Often the agent will waste a great deal of time in an inconsistent confused loop until it gives up or manages to stumble onto a state from which it can solve the problem. As a result, these statistics are misleading in that they tend to report repeatedly overestimations for the same inconsistent internal states.

The robustness of the Meliora's performance in the face of persistent overestimations led us to consider tasks with more than four blocks. Another set of experiments was performed in which the problem instances ranged from easy (0 blocks to unstack) to most difficult (3 blocks to unstack). In these experiments, however, additional outlying blocks were added to the pile. The number of outliers was randomly chosen between 0 and 20.[5] Outliers interfere with the system's ability to learn the most difficult instances because the agent's sensory motor sys-

[5]Subsequent experiments with as many as 50 blocks have shown similar results.

tem cannot distinguish between stacks containing four or more blocks. Therefore, the agent has no way of distinguishing (under any sensory-motor configuration) states where it has to unstack three blocks from states where it has to unstack 4, 5, 6, or more blocks. These states do not have consistent internal representations. Results from the experiments are shown in Figure 6.8. They are comparable to the results from the earlier experiment, except with slightly longer average solution times and a slightly lower success rate (especially for the most difficult instances). Nevertheless, even in the face of inconsistencies the agent is capable of learning a robust decision policy.

6.3 The Consistent Representation Method

The lion algorithm used by Meliora-II is a specific instance of a general technique which we call the *Consistent Representation (CR) Method.* The remainder of this chapter outlines the CR-method, describes the lion algorithm in terms of it, considers alternatives to the techniques used in Meliora II, and describes other recent work that makes use of the CR-method.

In the CR-method, control is comprised of two stages: an identification stage and an overt control stage (*cf.* Figure 6.2 and Figure 6.9). The objective of the identification stage is to generate a consistent, task dependent internal representation. This is accomplished by an identification procedure i that executes a series of non-invasive perceptual actions that collect the information needed to define a consistent internal state. Once a consistent internal state has been identified, an overt control procedure b is invoked which generates a single overt action. Both the identification and overt control procedures are adaptive. The identification procedure is adjusted to eliminate inconsistent states from the internal representation, and the overt control procedure is adjusted to maximize future expected return. Let u_i and u_b denote the procedures used to update the identification and overt control procedures, respectively. A schematic diagram of the CR architecture is shown in Figure 6.9.

The operations in the decision cycle of a CR-method are as follows:

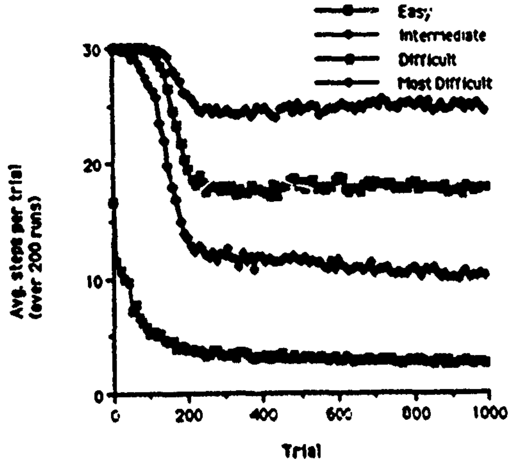
1. At time t, evaluate the identification procedure i, generating an internal state s'_t, which represents the current external state.

2. Using s'_t as input, evaluate the overt control procedure b, generating an overt action a'_t.

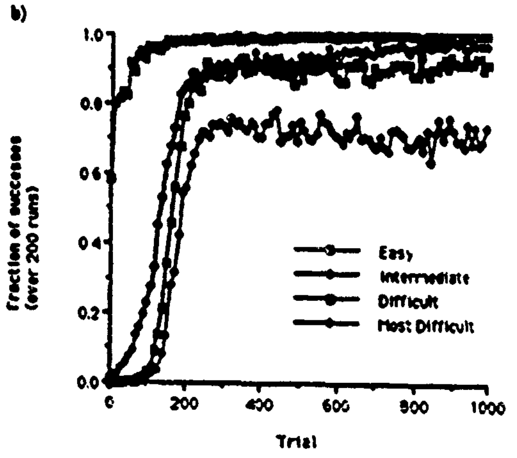
3. Perform a'_t and observe the next internal state s'_{t+1} and the reward r_t obtained.

4. Evaluate u_i, updating i based on the observations made in Step 3.

a)



b)



c)

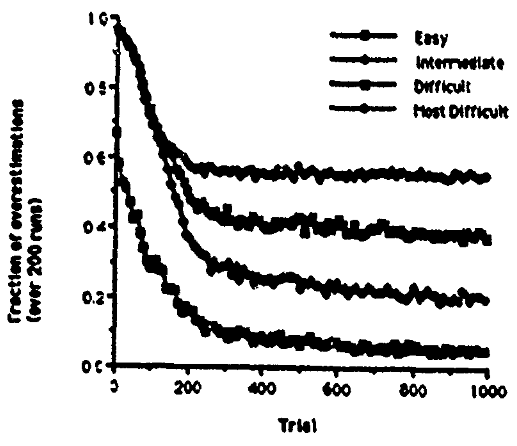


Figure 6.8: Performance plots for experiments that include piles of up to 20 outlying blocks. a) shows the average solution time for each class of problem, b) shows the success rate for each class of problem and c) shows the fraction of overestimations observed for each class of state. The plots are comparable to those in our original experiments and show that the agent can learn even in environments that it cannot consistently represent.

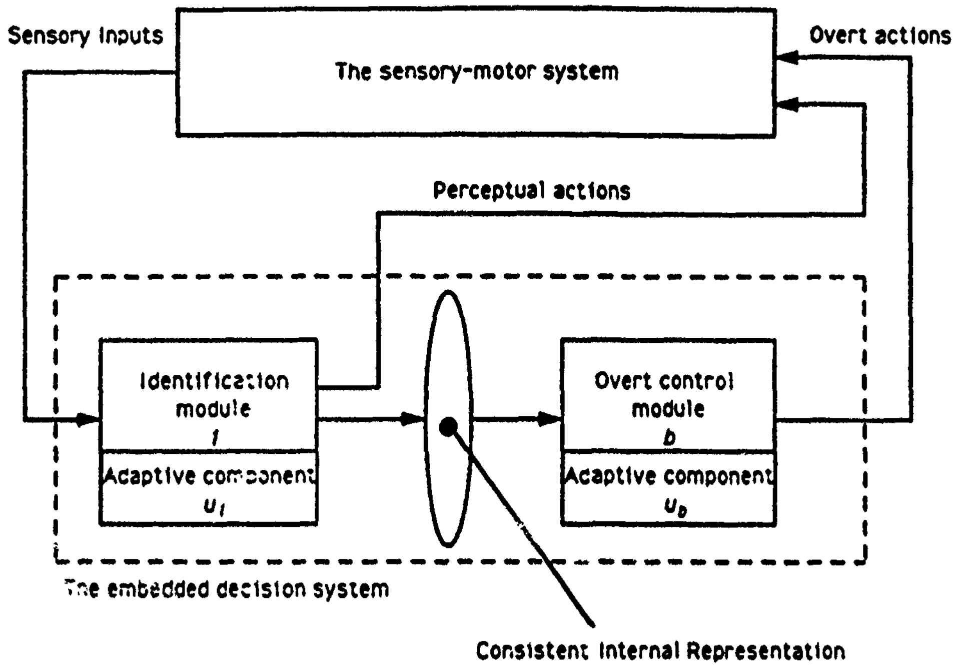


Figure 6.9: The basic architecture of a system using the CR-method. Control is accomplished in two stages: an identification stage, followed by an overt control stage. The goal of identification is to generate a consistent, task dependent internal state space. The goal of overt control is to maximize the future discounted return. In the figure, i and b represent the identification and overt control procedures, respectively. i and b are both adaptable, and the algorithms used to update them are u_i and u_b, respectively.

5. Evaluate u_b, updating b based on the observations made in Step 3.

The central idea behind the CR-method is that the agent aims to learn and base its actions on an internal representation of the task that is consistent. Each internal state is assumed to define an equivalence class of external states that all share the same set of action-values. In other words, each internal state satisfies the Markov property with respect to predicting future rewards — information in addition to the internal state does not improve the agent's ability to predict future reward. When inconsistencies exist, they are detected and eliminated. This consistency assumption is derived from our desire to use existing reinforcement learning techniques for overt control. That is, if overt control is to be learned using Q-learning [Watkins, 1989], the AHC algorithm [Sutton, 1984], the bucket brigade [Holland *et al.*, 1986], or other learning algorithms based on temporal differences, then the internal representation generated by the identification stage must be consistent. Our desire to use reinforcement learning for overt control ***constrains*** the form of the internal decision problem and ***defines the requirements*** for the identification stage. It also imposes requirements on the sensory-motor system. In particular, the sensory system must be capable of providing at each point in time enough information to identify a consistent internal state. If sufficient information cannot be attained from immediate sensor data, then the sensory system may need to be augmented with some kind of short term memory that can be used to keep track of relevant information from the past.

One might say that with the CR-method we are aiming to construct internal decision problems that are Markov. While it is the case that a Markov internal decision problem has a state space that is consistent, it need not be the case that a consistent internal state space defines a Markov decision problem. In particular, it is possible for an internal decision problem to have a consistent internal state space but have a non-Markov transition function. Thus, the class of decision problems with consistent state spaces is a strict superset of the class of Markov decision problems. Nevertheless, it is intuitively correct to think of an agent as learning a Markov representation of the external task.

6.3.1 Meliora-II as a CR system

Meliora-II, and the lion algorithm it employs, is one instantiation of the CR-method. However, there are many other ways to implement the various components of the architecture and these are worth exploring. In this subsection, we discuss Meliora-II from the general perspective of the CR-model and discuss a number of alternatives. In the next subsection, we briefly describe two other systems [Chapman and Kaelbling, 1991; Tan, 1991a; Tan, 1991b] that also exemplify the CR-method.

Overt Control in Meliora-II

From the description in Figure 6.1, it may not be completely apparent that Meliora-II uses 1-step Q-learning for adaptive overt control. However, notice that at each time step a single internal state is identified (the lion) and used to guide decision making and utility estimation. When the lion states do not overestimate (i.e., are not suspected of being inconsistent) their action-values are updated using the 1-step Q-learning rule. Also, notice that the utility estimates used in the updating rule are just the utility estimates of the lions (the identified states). Updating the action-values for non-lions also uses a modified 1-step estimator, but is better thought of as part of the identification process. Thus in Meliora-II, u_b, the updating algorithm for overt control, corresponds to the updating procedure used in 1-step Q-learning.

Similarly, the overt control procedure, b, used in Meliora-II corresponds to a simple version of 1-step Q-learning. Namely, with fixed probability p the agent chooses an action at random; otherwise, it performs the action with the largest action-value.

Alternative Approaches to Overt Control

In general, any number of reinforcement learning algorithms can be used to implement overt control. We used 1-step Q-learning because it is simple and convenient, but other techniques, such as multi-step Q-learning [Watkins, 1989], AHC algorithms [Sutton, 1984], bucket brigade algorithms [Holland *et al.*, 1986], and Interval Estimation algorithms [Kaelbling, 1990], can be used as well.

Identification in Meliora-II

In Meliora-II, the identification procedure proceeds by executing a series of perceptual actions which collect a set of candidate internal representations. The perceptual actions executed are determined by a perceptual control policy which maps internal states into perceptual actions. This policy is learned using elements of 1-step Q-learning. However, instead of estimating future returns, the action-values encode information about the likelihood of generating consistent internal states. Perceptual actions that cause the system to focus on relevant aspects in the environment and lead to consistent internal states tend to have larger action-values than perceptual actions that focus attention on irrelevant aspects of the environment. Once a number of candidate internal states are collected, the state with the largest action-value is selected to represent the current external state.

A crucial operation in any algorithm based on the CR-method is the identification of inconsistent internal states. In Meliora-II, inconsistent internal states are detected by monitoring the sign in the estimation error of the 1-step Q-learning

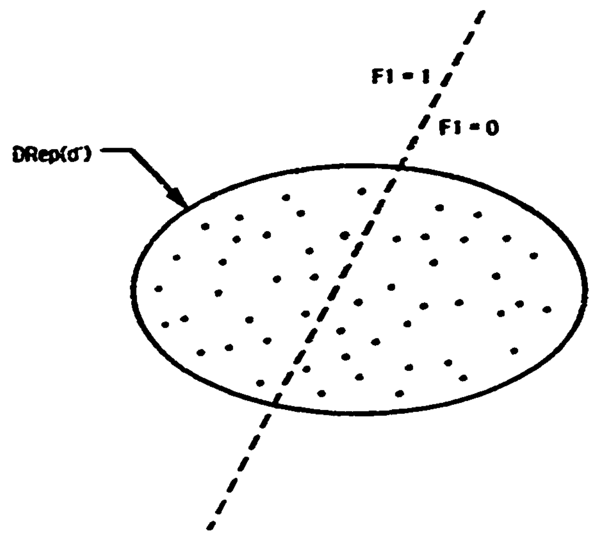


Figure 6.11: Inconsistent decisions can be detected by partitioning the equivalence class of a decision in two and looking for differences in the return distributions for the two subsets. If a partitioning exists for which the expected returns for the two subsets differ, then the decision is necessarily inconsistent. If no such partitioning exists, then the decision is consistent.

the way until a leaf is encountered. If every leaf node in the tree is consistent, then the tree represents a universal identification procedure. If not, the tree can be refined by detecting the inconsistent leaves and refining them by splitting them with additional sensing operations.

In Meliora-II inconsistent states are detected based on overestimation of the action-value function. While this technique is sufficient for the GB-task, it is extremely limited since it works for decision problems that are deterministic and that have non-negative rewards only. Fortunately, there are other techniques that are more general and probably more effective in the long run.

By definition, an internal decision is inconsistent if any of the decisions it represents in the external model have different optimal action-values. Thus, one way to determine whether or not a decision d' is inconsistent is to split its equivalence class, *DRep(d')*, into two or more subsets and look for differences in the expected returns of each subset. If a partitioning exists such that the expected return for two subsets differ, then the decision is necessarily inconsistent. If no such partitioning exists then the decision is consistent. This idea is illustrated graphically in Figure 6.11. The basic idea is to keep track of some extra information (e.g., previous state information or additional sensory input bits) that can be used to partition *DRep(d')* and then use statistical techniques to determine if the expected returns for these subsets are identical. There are several statistical methods

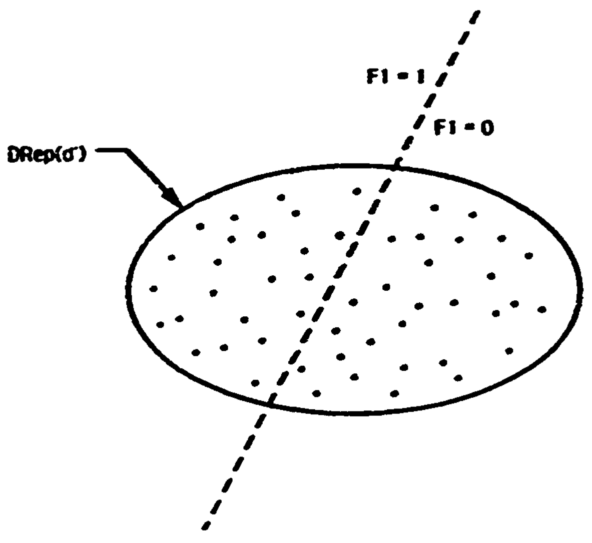


Figure 6.11: Inconsistent decisions can be detected by partitioning the equivalence class of a decision in two and looking for differences in the return distributions for the two subsets. If a partitioning exists for which the expected returns for the two subsets differ, then the decision is necessarily inconsistent. If no such partitioning exists, then the decision is consistent.

the way until a leaf is encountered. If every leaf node in the tree is consistent, then the tree represents a universal identification procedure. If not, the tree can be refined by detecting the inconsistent leaves and refining them by splitting them with additional sensing operations.

In Meliora-II inconsistent states are detected based on overestimation of the action-value function. While this technique is sufficient for the GB-task, it is extremely limited since it works for decision problems that are deterministic and that have non-negative rewards only. Fortunately, there are other techniques that are more general and probably more effective in the long run.

By definition, an internal decision is inconsistent if any of the decisions it represents in the external model have different optimal action-values. Thus, one way to determine whether or not a decision d' is inconsistent is to split its equivalence class, *DRep(d')*, into two or more subsets and look for differences in the expected returns of each subset. If a partitioning exists such that the expected return for two subsets differ, then the decision is necessarily inconsistent. If no such partitioning exists then the decision is consistent. This idea is illustrated graphically in Figure 6.11. The basic idea is to keep track of some extra information (e.g., previous state information or additional sensory input bits) that can be used to partition *DRep(d')* and then use statistical techniques to determine if the expected returns for these subsets are identical. There are several statistical methods that can be used to determine whether or not the expected returns are identical. The student's T-test [Lehmann, 1959] used by Chapman and Kaelbling [Chapman and Kaelbling, 1991] is perhaps the most appropriate since in many cases the return distribution is likely to be approximately normal. However, other parametric and nonparametric tests [Bradley, 1968; Randles and Wolfe, 1979] may also prove useful, including techniques based on uncertainty intervals [Kyburg, 1991; Kaelbling, 1990].

The appeal of statistical methods is that they apply in general. However, one troublesome assumption made by statistical methods is that the underlying stochastic process is stationary. Unfortunately, in reinforcement learning the return distribution is almost never stationary since the agent is continually updating its utility estimates and its control policy. Thus, detecting statistical differences in return distributions may be troublesome.[6]

In general for any random variable X, we will say that the internal decision $d' = (s'a')$ has uniform statistics (or is consistent) with respect to X if any and all subsets of $DRep(d')$ have the same underlying distribution with respect to X. Given this definition, a related statistical approach that may be useful for circumventing the non-stationarity associated with return distributions is to detect inconsistent states by enforcing *local consistency* requirements. In particular, if each internal decision is consistent with respect to the next internal state and the next immediate reward, then every decision is necessarily consistent with respect to the return. Since these local statistics (next state and immediate reward) are stationary, they are likely to be easier to estimate and compare accurately. This local approach is equivalent to monitoring the internal decision process to ensure that it is Markov. When a statistical test shows a state to be non-Markov with respect to immediate rewards and predicting the transition function, it is inconsistent and the identification function must be refined.

A third approach to detecting inconsistent states that avoids statistical techniques altogether is to rely on feedback from external sources. For instance, if the learning agent is embedded in a cooperative social environment where it can receive immediate feedback from an external supervisor or can watch the behavior of another skilled agent, then it may be possible for the agent to deduce which states are inconsistent by detecting variation in the feedback or behavior of the external agent. For instance, if an external critic provides a signal indicating the correctness of an action immediately and if the signal varies for a given internal decision, then the decision must be inconsistent (assuming the external critic is reliable). Similarly, if for a given internal state the agent observes a variation in the actions performed by a skilled role model, then the state is probably inconsis-

[6]Chapman and Kaelbling [Chapman and Kaelbling, 1991] have successfully used the Student's T-test on return distributions to detect inconsistencies for a relatively complex task. Judging from their success, non-stationarity may be less of a problem than expected. Certainly more empirical evidence is needed to know for sure.

tent (again assuming the role model's behavior is reliably optimal). These more direct methods for detecting inconsistent states make strong assumptions about the environment in which the agent is embedded, but can substantially reduce the time needed to learn consistent internal representations. We have successfully applied these direct methods in the versions of Meliora described in Chapter 7.

6.3.2 Other CR-systems

Two other adaptive control algorithms that are excellent examples of the CR-method are the G-algorithm [Chapman and Kaelbling, 1991] and the CS-QL algorithm [Tan, 1991a; Tan, 1991b].

The G Algorithm

Recent work by Chapman and Kaelbling aims to address the problem of having to generalize over state spaces generated by sensory inputs with excessive irrelevant detail. In their target domain the sensory system generates more than one hundred bits of input, most of it irrelevant. While the large input vector defines a consistent internal representation, the overwhelming size of the internal state space that results ($\approx 2^{100}$ states) severely interferes with Q-learning by making too many distinctions. They call the problem of filtering out the irrelevant information the *input generalization problem.* Their approach, called the G-algorithm, is to incrementally build a decision (or classification) tree, called a G-tree. The G-tree partitions the set of possible inputs into a much smaller set of internal states. The leaves of the tree define the internal states and internal nodes identify relevant input bits (or tests) to perform during identification. The tree represents a universal identification procedure in which, at each time step, the input vector is classified by traversing the tree from the root to a leaf by following the branches whose values match the bits in the input vector. Input bits that are not explicitly tested in the G-tree are considered irrelevant and are ignored.

The objective of the G-tree is to define an internal state space that is consistent. Internal states that are not consistent are detected and split into two new states by adding relevant bit tests to the leaves in the tree. To detect inconsistent states the G-algorithm uses the Student's T-test on two statistics: the immediate reward and the discounted future return. If dividing a leaf node in two (by adding another input bit to the tree) leads to subnodes that are statistically different, then the leaf is suspected of being inconsistent and split along that bit.

The G-algorithm is used to learn an identification procedure (i.e., the G-tree), while another component learns an overt control policy. For overt control, Chapman and Kaelbling use a variation on Q-learning called the Interval Estimation (IE) algorithm [Kaelbling, 1990]. Using the IE algorithm along with the

G-algorithm, they have demonstrated a system that learns an efficient G-tree and learns to solve a difficult sequential decision problem.

The CS-QL Algorithm

A similar system was independently developed by Tan [Tan, 1991a; Tan, 1991b]. Like Chapman and Kaelbling's, Tan's system uses an incrementally built decision tree to classify situations into internal states. However, Tan's algorithm, called CS-QL, is different in several ways. First, instead of splitting nodes by bits in the sensory input, nodes are split on more general sensing operations that have costs and may be multi-valued. Second, insertion of sensing operations into the decision tree takes into account both the *utility* and the *cost* of the operation, thus yielding a *cost-sensitive* decision tree. This is an advancement over the G-algorithm, which does not account for the cost of perceptual actions. In the lion algorithm, cost-sensitive perception is achieved by discounting the rewards used to update the perceptual action-value function. The CS-QL algorithm uses an overestimation technique similar to the one used in Meliora-II to detect inconsistent states.

For overt control, the CS-QL algorithm uses 1-step Q-learning. Since inconsistent states are detected using the overestimation, CS-QL is restricted to deterministic tasks with non-negative rewards. Nevertheless, Tan has demonstrated the algorithm in a system that efficiently learns to identify landmarks and navigate in a simulated 2-D environment.

7 Cooperative Mechanisms

7.1 Introduction

Reinforcement learning involves a process that searches the world for states that yield reward. But for most real-world tasks, the state space is large and rewards are sparse. Under these circumstances the time required to learn an adequate control policy can be excessive. The detrimental effects of search manifest themselves most at the beginning of the task, when lack of knowledge can lead to unbiased random search, and in the middle of a task, when changes in the environment invalidate an existing control policy.

In nature, intelligent agents do not exist in isolation but are embedded in a benevolent society that guides and structures learning. Humans learn in rich, carefully structured environments; they learn by watching others, by being told, and by receiving criticism and encouragement. *Learning is more often a transfer than a discovery.* Similarly, robots cannot be expected to learn very much in isolation. They must be embedded in cooperative environments, and algorithms must be developed to facilitate the transfer of knowledge among them. Within this context, the reinforcement learning framework continues to play a vital role:

1. for pure discovery purposes — that is, reinforcement learning can be useful for increasing the collective knowledge of a society as a whole,
2. for refining and elaborating knowledge gained from others,
3. for carrying on in the absence of guidance from others and for interpolating between periods of interaction, and
4. for providing a simple signaling mechanism (rewards) for communicating and transferring knowledge.

This chapter proposes two cooperative mechanisms to reduce search and decouple the learning rate from state-space size. The first approach, called *Learning* *with an External Critic* (LEC), is based on the idea of a mentor who watches the learner and generates rewards, which signal the correctness of the agent's most recent action. This reward is used temporarily to bias the learner's overt control strategy and to detect inconsistent internal states more quickly. The second approach, called *Learning By Watching* (LBW), is based on the idea that an agent can gain valuable experience vicariously by relating the observed behavior of others to its own. While LEC techniques require interaction with an external agent that is knowledgeable and attentive, LBW techniques can be effective even when the external agent is unskilled and unaware of the learner.

The chapter begins by developing and demonstrating LEC and LBW algorithms in the context of the GB-task. Two new programs, Meliora-III-LEC and Meliora-III-LBW are described and shown to outperform Meliora-II significantly. A formal analysis of search is then presented. To facilitate the analysis, attention is focused on a restricted (but representative) class of decision problems, called *homogeneous problem solving tasks.* For these tasks, reinforcement learning algorithms that rely on random walks to solve problems initially are shown to have expected learning times that are at least exponential in the depth of the state space.[1] For Q-learning, true random walks can be avoided by proper selection of an initial (unbiased) action-value function. In this case, the expected learning time appears to be at least polynomial in the state space depth. Further improvement can be made by using the LEC and LBW algorithms, which have expected learning times that are at most linear in the size of the state space and, under appropriate conditions, are independent of the state space size altogether and proportional to the length of the optimal solution path.

7.2 Learning with an External Critic

Learning with an external critic (LEC) enhances the learning environment by providing helpful hints that indicate the appropriateness of the robot's most recent actions. LEC algorithms achieve faster learning by eliminating the delay between the performance of an action and its evaluation (feedback), thus facilitating credit assignment. Depending upon the communication skills of the learner and the critic, a range of LEC algorithms can be devised [Whitehead and Ballard, 1991b]. A particularly simple approach, called *Binary LEC* (B-LEC) is to assume that after each overt action an external critic generates a binary signal, $sig(t)$, with probability p_{critic} according to the rule:

$$sig(t) = \begin{cases} \text{YES} & \text{if } a_t \text{ is optimal} \\ \text{NO} & \text{otherwise} \end{cases} \tag{7.1}$$

[1]The *depth* of a state space is formally defined in Section 7.4; however, intuitively it corresponds to the maximum distance (measured in number of steps) between any two states in a state space.

where a_t is the overt action performed by the learner at time t.

To evaluate the potential utility of this immediate feedback a new program, Meliora-III-LEC, was developed to exploit this information for the GB-task. The control algorithm used by Meliora-III-LEC is called the *Biasing Binary LEC* (BB-LEC) algorithm.

7.2.1 Meliora-III-LEC

In Meliora-III-LEC, immediate feedback from the critic is used to facilitate adaptation in both overt control and state identification. With respect to overt control, the critic's signal is used to bias the robot's overt control policy. This is especially useful during the early stages of learning since it reduces floundering. With respect to state identification, variation in the critic's feedback is used to detect inconsistent internal states quickly.

The decision algorithm used in Meliora-III-LEC is shown in Figure 7.1. Structurally, it is quite similar to the lion algorithm used by Meliora-II. However, a number of minor modifications have been made to take advantage of the information encoded in the external critic's signal.

Biasing Overt Control

In order to positively bias the robot's overt control policy, the critic's signal is converted into an internal source of reward. At time t, the robot generates an internal reward, $r_c(t)$, according to the rule:

$$r_c(t) = \begin{cases} +R_c & \text{if } sig(t) = \text{YES} \\ -R_c & \text{if } sig(t) = \text{NO} \\ 0 & \text{otherwise} \end{cases} \tag{7.2}$$

where R_c is a positive constant. According to this rule, positive and negative internal rewards are generated whenever positive or negative feedback is returned by the critic, respectively. When no signal is returned by the critic the internal reward is zero.

The reward received from the environment, denoted r_w, and the critic's reward are treated separately. Reward from the environment is treated as in Meliora-II — it is used to learn a specialized overt action-value function. Reward from the critic is used to learn a bias function B over internal state-action pairs. The bias function estimates the expected value of the immediate reward received from the critic for each state-action pair. In Meliora-III-LEC, the values of the bias function are updated using a simple temporally weighted average:

$$\forall_{s \in S_t} B_{t+1}(s_t, a_t) \leftarrow (1 - \psi) B_t(s_t, a_t) + \psi r_c(t), \tag{7.3}$$

Overt Cycle:

```
1) Execute Perceptual Cycle and generate $S_t$, a set of internal
   representations for the current world state.
2) Let $d_t = (s_t, a_t)$ be the maximal internal decision:
   $d_t = \arg\max_{(s,a) \in S_t \times A_O} [Q_I(s,a) + B(s,a)]$
3) Choose a state, the lion, to represent the current world state: $lion = s_t$.
4) Estimate the utility of the current world state, $s_t$: $V_E(s_t) \leftarrow V_I(lion)$.
5) Execute Update-Overt-Estimates based on $V_E(s_t)$, $r_{t-1}$, $oact_{t-1}$, $lion_{t-1}$, $sig(t-1)$;
   where $r_t$ is the reward received at time $t$, $oact_{t-1}$ is the last overt action executed,
   $lion_{t-1}$ is the internal state selected to represent the previous world state,
   and $sig(t-1)$ is the critic's signal for time $t-1$.
6) Choose the next overt action to execute:
   With probability $p$ follow policy $f_o(lion) = a_t$,
   Otherwise choose randomly: $oact \leftarrow Random(A_O)$
7) Execute $oact$ to obtain $r_t$, $sig(t)$, and $s_{t+1}$.
8) Go to 1).
```

Update-Overt-Estimates:

```
1) Estimate the error in the lion's action-value: $E_{lion} \leftarrow (r_{t-1} + \gamma V_E(s_t)) - Q_I(lion_{t-1}, oact_{t-1})$.
2) Update the action-value of the lion:
   If ($E_{lion} < 0$) or
     ($B(lion_{t-1}, a_{t-1}) > 0$ and $sig(t-1) = $ NO) or ($B(lion_{t-1}, a_{t-1}) < 0$ and $sig(t-1) = $ YES)
   then the lion is suspected of being inconsistent, so suppress it: $Q_I(lion_{t-1}, oact_{t-1}) \leftarrow 0.0$
   Else update it using the standard 1-step Q-learning rule:
     $Q_I(lion_{t-1}, oact_{t-1}) \leftarrow Q_I(lion_{t-1}, oact_{t-1}) + \alpha E_{lion}$.
3) Update non-lion internal states:
   For each $s \in S_{t-1}$ and $s \neq lion_{t-1}$ do:
     Let $E_s = r_{t-1} + \gamma V_E(s_t) - Q_I(s, oact_{t-1})$
     If ($E_s < 0$) or
       ($B(s, a_{t-1}) > 0$ and $sig(t-1) = $ NO) or ($B(s, a_{t-1}) < 0$ and $sig(t-1) = $ YES)
     then $s$ is suspected of being inconsistent, so suppress it: $Q_I(s, oact_{t-1}) \leftarrow 0.0$
     Else update it using the lion's error:
       $Q_I(s, oact_{t-1}) \leftarrow Q_I(s, oact_{t-1}) + \alpha' E_{lion}$ — where $\alpha' < \alpha$.
4) Update the bias function $B$:
   For each $s \in S_{t-1}$ do: $B(s, a_{t-1}) \leftarrow (1 - \psi)(s, a_{t-1}) + \psi r_c(t-1)$.
```

Figure 7.1: An outline of the decision procedure implemented by Meliora III-LEC. This procedure is similar to the lion algorithm, except that reward from the critic is 1) used to learn a bias function B, which is used to bias the agent's overt control policy, and 2) used to detect inconsistent internal states. The **Perceptual Cycle** is not shown here since it is identical to the one described in Figure 6.1.

where S_t is the set of candidate internal states returned by the perceptual cycle and a_t is the internal overt action executed at time t. The learning rate parameter, ψ, is a constant between 0 and 1. The average is temporally weighted so that the robot can "forget" old advice that has not been recently repeated. Without it, the robot has difficulty adapting to changes in the task once advice is extinguished or changed.

The decision rule for overt control is simple. At time t, let $d_l = (s_l, a_l)$ be the candidate internal decision that maximizes the sum of the action-value and the bias function. That is,

$$d_l = \arg\max_{(s,a)\in S_t\times A_O}[Q_I(s,a) + B(s,a)]. \tag{7.4}$$

In Meliora-III, the state identified to represent the current external world, the *lion*, is simply the state, s_l, of the maximal decision, d_l. Similarly, the robot's overt policy, $f_o(S_t)$, is the action a_l of the maximal decision. Using this rule, decisions that have previously been associated with positive feedback from the critic are preferred over decisions that have received no feedback, and decisions associated with negative feedback tend to be suppressed.

After executing an action and observing the reward, next state, and critic's signal that result, the action-value function, Q_I, is updated more or less the same as in Meliora-II. If a decision is suspected of being inconsistent it is suppressed. Otherwise, if it is the lion's decision, it is updated using the 1-step Q-learning rule; if not, it is a non-lion and is updated based on the lion's error. The bias function is also updated at this time.

The discounted return and the critic's immediate reward are estimated separately to ensure that subsequent extinction of the critic's feedback does not inadvertently disrupt learning of the underlying decision problem. If the two rewards are combined and used to estimate a single action-value function, then subsequent extinction of the critic's feedback leads to a reduction in the overall expected return and errors in the action-value function. These errors, in turn, cause a prolonged period of non-optimal behavior, while the agent estimates a new action-value function for the original, underlying decision problem (see [Whitehead and Ballard, 1991b] for details). By separating the two rewards, the robot is able to learn the action-value function for the underlying decision problem directly. In this case, the only effect of feedback extinction is on the bias function, whose values gradually decrease to zero. This allows the external critic to terminate "programming" once the agent has learned the task.

Detecting Inconsistent Decisions

In Meliora-III-LEC, inconsistent decisions are detected both by using the overestimation technique described previously and by using feedback from the external critic. This combined approach leads to improved performance when the critic is available but also allows the robot to fall back on an effective, but slower, method when immediate feedback is absent.

To detect inconsistent decisions using the critic, the agent simply looks for internal state-action pairs that have resulted in contradicting signals. That is, if executing the decision $d = (s, a)$ yields positive feedback at one point in time and negative feedback at another, then (assuming the critic is reliable and the task is stationary) it must be inconsistent.

Instead of keeping track of all the signals ever received for each internal decision, the biasing function is used to detect contradicting signals. In particular, at time t, for each $s \in S_{t-1}$, the decision $d \in (s, a_{t-1})$ is considered inconsistent if either

$$B(s, a_{t-1}) > 0 \quad \text{and} \quad sig(t-1) = \text{NO}$$

or

$$B(s, a_{t-1}) < 0 \quad \text{and} \quad sig(t-1) = \text{YES}.$$

In the first case, the positive value of the bias function indicates that in the past positive signals have been received for this decision, which contradicts the current signal. In the second case, the negative bias value indicates previous negative signals, contradicting the present positive signal. Both cases indicate a contradiction.

7.2.2 Experimental Results

Meliora-III-LEC was applied to the GB-task to test the BB-LEC algorithm and to determine the effect of an external critic on the learning rate. Each experiment consisted of 100 runs of 250 trials each. To demonstrate the degree of thrashing that occurs in an an unsupervised system, the time limit, n_{quit}, was increased to 100 steps. Plots of the average solution time versus trial number are shown in Figure 7.2. The results shown in this figure are for $\gamma = 0.8$, $\psi = 0.6$, $\alpha = 0.2$, $p = 0.9$, $p' = 0.9$, and $R_c = 500$. Qualitatively similar results were obtained for a wide range of parameter values. Each curve in the figure corresponds to a different rate of feedback from the external critic, ranging from no feedback ($p_{critic} = 0.0$, Meliora-II) to feedback after each overt action ($p_{critic} = 1.0$). The figure clearly demonstrates the performance improvement gained by addition of an external critic. Even occasional feedback (e.g., $p_{critic} = 0.2$) has a tremendous effect.

To assess the effect of the critic's signal on state identification, the average number of suppressed decisions per trial is plotted versus trial number for the GB-task in Figure 7.3. During the first few trials, the systems that receive immediate feedback begin to learn the bias function and have high suppression rates. Some

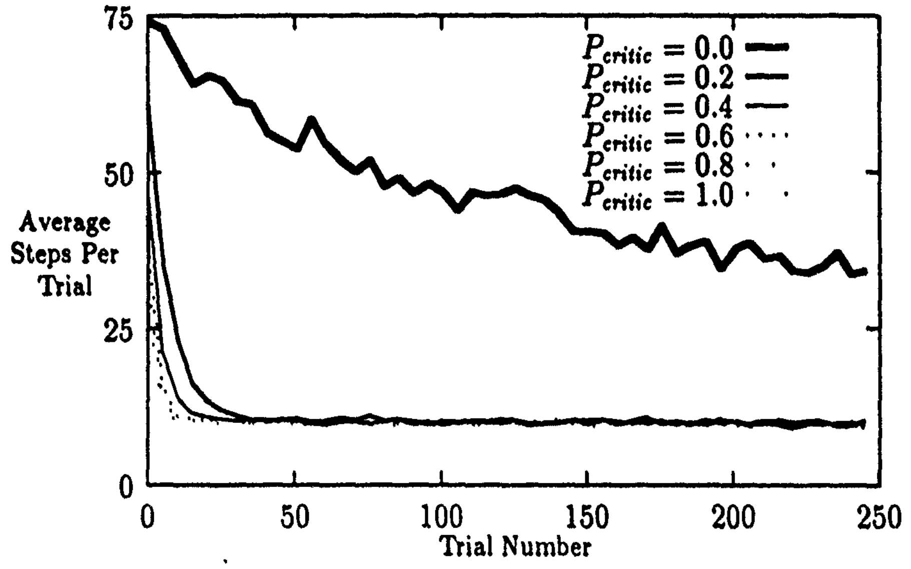


Figure 7.2: Plots of the average solution time versus trial number for Meliora-III-LEC for p_{critic} in the range [0.0, 1.0]. The plots show that feedback from an external critic is extremely useful for improving the learning rate, even when it occurs only occasionally.

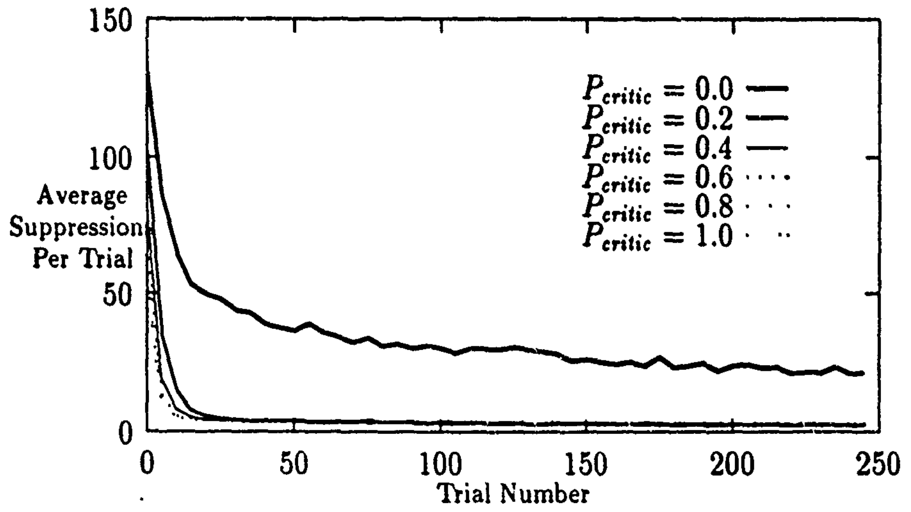


Figure 7 3: The average (over 100 runs) of the number of suppressed decisions per trial of Meliora-III-LEC for $p_{critic} \in [0.0, 1.0]$.

of these suppressions are due to contradictions in the critic's signal; however, most are due to action-value overestimations since in these experiments all action-values were initialized to 1.0. After a dozen or so trials the systems with feedback begin to perform nearly optimally and the suppression rate drops dramatically. The low steady state suppression rate can be explained by noting that inconsistent decisions tend to have negative (or lower) bias values. This tends to keep them from competing for lionhood when consistent states are spuriously suppressed in the steady state.

In a second experiment, the task was changed after 250 trials so that the robot was rewarded for picking up a red block (instead of a green one). Most reinforcement learning algorithms (e.g., Q-learning) have a difficult time adapting to such drastic changes since the previously learned action-value function interferes with learning the new task by strongly biasing the agent's behavior away from the new source of reward. Moreover, the time needed to "unlearn" (or change) the existing action-value function is substantial since changes to action-values are incremental [Whitehead and Ballard, 1991b]. However, as shown in Figure 7.4, Meliora-III-LEC has little trouble adapting to the change. The incorrect existing policy does not significantly interfere with learning in Meliora since quick suppression of inconsistent decisions makes "unlearning" almost immediate. Moreover, the suppression rate at the point of change (trial 250) is lower than the initial suppression rate. This follows since most of the useless decisions (i.e., those that leave the external state unchanged) have already been detected as inconsistent and suppressed by trial 250.

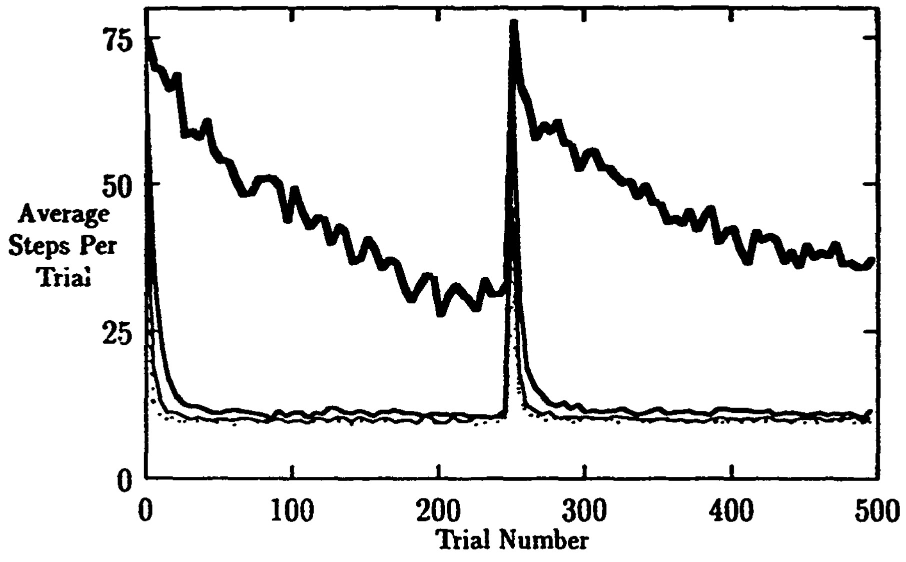
7.3 Learning By Watching

Another technique that can be used to improve the learning rate is to gain additional experience by observing and interpreting the behavior of others. We call this approach *Learning-By-Watching* (LBW).

The conceptual organization of an agent using LBW is shown in Figure 7.5. The agent has two fundamental modes: a performance mode and a watching mode. In the performance mode, the agent acts like any other adaptive system (i.e., it observes the state, chooses and executes an action, observes the outcome, and adapts its policy accordingly). In the watching mode, the situation is similar except the agent uses the behavior of another agent as its source of experience. Depending upon the mode of operation, different sensing/behavior-interpretation hardware is used to generate the state-action-reward sequences used for learning.

In our work with LBW we have assumed that the learner can correctly recognize the state-action-reward triple of any agent it is observing. This sequence is then used for learning just as if it were the agent's own personal experience.

a)



b)

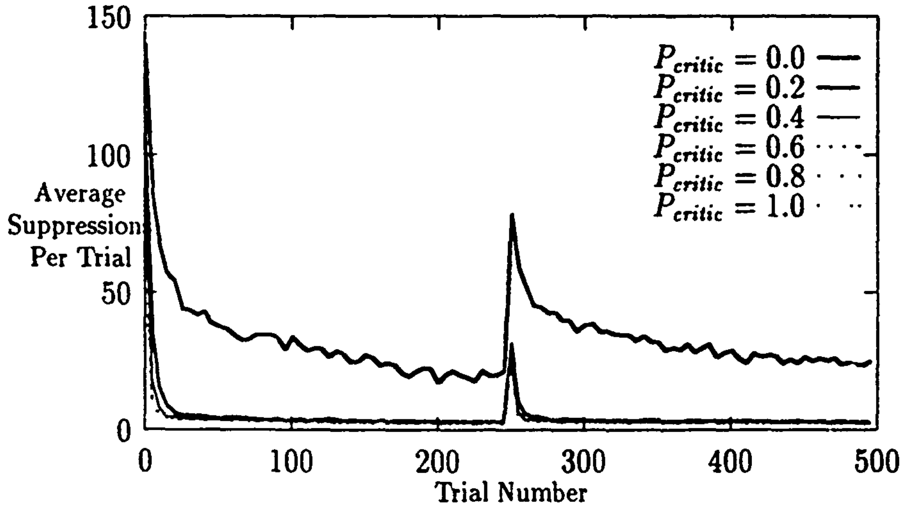


Figure 7.4: Performance of Meliora-III-LEC in response to a change in the underlying decision problem. In this experiment the robot receives a reward for picking up a green block for the first 250 trials and a reward for picking up a red block for the last 250 trials. Quick suppression of inconsistent decisions allows Meliora-III to adapt to task changes faster than more incremental approaches.

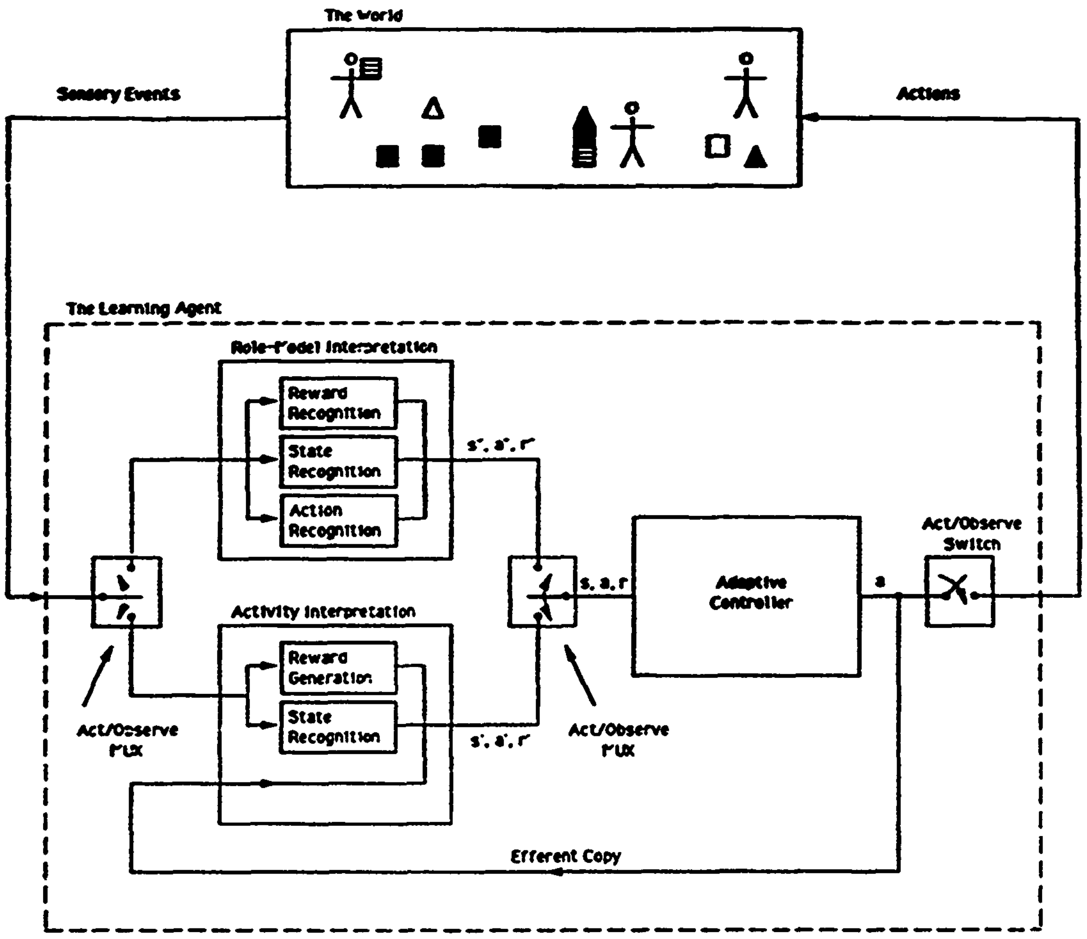


Figure 7.5: The conceptual organization of an agent using LBW. In this architecture, observations of other agents are used as an alternative source of experience for the embedded learner.

Although this assumption is overly simplistic and ignores many important issues, it is reasonable considering our objective — to explore the potential benefits of integrating reinforcement learning with more powerful cooperative mechanisms like "learning-by-watching." The general issue of recognizing and interpreting the behavior of others is fundamental to these cooperative mechanisms, and some promising early work in this area has been reported [Newtson *et al.*, 1977; Tsuji *et al.*, 1977; Thibadeau, 1986; Hirai and Sato, 1989; Kautz, 1987; Kuniyoshi *et al.*, 1990]. However, many important problems remain unaddressed.

Depending upon the particular circumstances of the agent and the environment in which it is embedded, a number of LBW algorithms can be devised and a range of performance improvements attained. A learner capable of observing a group of equally naive peers solving similar tasks can gain from its observations, but not nearly as well as when it observes a skilled role model. Further, if a learner "knows" that the agent it observes is skilled, it can exploit that knowledge to make even better use of its observations. We have studied a range of LBW algorithms under a variety of conditions in a simple simulated environment [Whitehead and Ballard, 1991b]. The results of these experiments indicate that LBW techniques are robust and effective for a wide range of learning settings. With respect to Meliora and the block-stacking task, we have only studied LBW in the context of a skilled role model that is known to perform optimally. An LBW algorithm for exploiting this information and experimental results for it on the GB-task are described below.

7.3.1 Meliora-III-LBW

Meliora-III-LBW is a program that learns the GB-task with the help of LBW. The program uses a version of the lion algorithm that has been modified to accommodate observations of a skilled role model. In this case, the embedded controller uses two streams of experience for learning: personal experiences and observed experiences. Personal experiences are used for adaptive perception and action much as in Meliora-II. Observed experiences are treated somewhat differently. With respect to overt control, observed experiences are used to learn a bias function B over state-action pairs. This bias function is used to bias overt control just like in LEC. With respect to state identification, observed experiences are used to identify potentially inconsistent states more quickly.

There are two modes of operation in Meliora-III-LBW: a performance mode and a watching mode. The control algorithm used during the performance mode is shown in Figure 7.6. This algorithm is almost identical to the lion algorithm used by Meliora-II (Figure 6.1), except that the choices of the lion and the policy action are affected by the bias function. The control algorithm used while in the watching mode is shown in Figure 7.7. In this mode, each control cycle begins by performing the perceptual function to collect a sequence of candidate

Overt Cycle:

```
1) Execute Perceptual Cycle and generate $S_t$, a set of internal
   representations for the current world state.
2) Let $d_l = (s_l, a_l)$ be the maximal internal decision:
   $d_l = \arg\max_{(s,a) \in S_t \times A_O} [Q_I(s,a) + B(s,a)]$
3) Choose a state, the lion, to represent the current world state: $lion = s_l$.
4) Estimate the utility of the current world state, $s_t$: $V_E(s_t) \leftarrow V_I(lion)$.
5) Execute Update-Overt-Q-Estimates based on $V_E(s_t)$, $r_{t-1}$, $oact_{t-1}$, and $lion_{t-1}$;
   where $r_t$ is the reward received at time $t$, $oact_{t-1}$ is the last overt action executed,
   and $lion_{t-1}$ is the internal state selected to represent the previous world state,
6) Choose the next overt action to execute:
   With probability $p$ follow policy $f_o(lion) = a_l$,
   Otherwise choose randomly: $oact \leftarrow Random(A_O)$
7) Execute $oact$ to obtain a reward $r_t$, and a new world state, $s_{t+1}$.
8) Go to 1).
```

Update-Overt-Q-Estimates:

```
1) Estimate the error in the lion's action-value:
   $E_{lion} \leftarrow (r_{t-1} + \gamma V_E(s_t)) - Q_I(lion_{t-1}, oact_{t-1})$.
2) Update the action-value of the lion:
   If ($E_{lion} < 0$)
   Then the lion is suspected of being inconsistent, so suppress it:
     $Q_I(lion_{t-1}, oact_{t-1}) \leftarrow 0.0$
   Else update it using the standard 1-step Q-learning rule:
     $Q_I(lion_{t-1}, oact_{t-1}) \leftarrow Q_I(lion_{t-1}, oact_{t-1}) + \alpha E_{lion}$.
3) Update non-lion internal states:
   For each $s \in S_{t-1}$ and $s \neq lion_{t-1}$ do:
     Let $E_s = r_{t-1} + \gamma V_E(s_t) - Q_I(s, oact_{t-1})$
     If ($E_s < 0$)
     Then $s$ is suspected of being inconsistent, so suppress it: $Q_I(s, oact_{t-1}) \leftarrow 0.0$
     Else update it using the lion's error:
       $Q_I(s, oact_{t-1}) \leftarrow Q_I(s, oact_{t-1}) + \alpha' E_{lion}$ — where $\alpha' < \alpha$.
```

Figure 7.6: An outline of the decision procedure used by Meliora III-LBW in the performance mode. This procedure is similar to the lion algorithm used by Meliora II, except the control policy is biased by observations made during the watching mode. The **Perceptual Cycle** is not shown here since it is identical to the one used in Meliora II.

Overt Cycle:

```
1) Execute Perceptual Cycle and generate $S_t$, a set of internal
   representations for the current world state.
2) Let $d_l = (s_l, a_l)$ be the maximal internal decision:
   $d_l = \arg\max_{(s,a) \in S_t \times A_O} [Q_I(s,a) + B(s,a)]$
3) Choose a state, the lion, to represent the current world state: $lion = s_l$.
4) Estimate the utility of the current world state, $s_t$: $V_E(s_t) \leftarrow V_I(lion)$.
5) Execute Update-Overt-Estimates based on $V_E(s_t)$, $r_{t-1}$, $a^o_{t-1}$, and $lion_{t-1}$;
   where $r_t$ is the reward received at time $t$, $a^o_{t-1}$ is the last overt action observed,
   and $lion_{t-1}$ is the internal state selected to represent the previous world state,
6) Observe the overt action performed by the role model, $a^o_t$
7) Go to 1).
```

Update-Overt-Estimates:

```
1) Estimate the error in the lion's action-value:
   $E_{lion} \leftarrow (r_{t-1} + \gamma V_E(s_t)) - Q_I(lion_{t-1}, oact_{t-1})$.
2) Update the action-value of the lion:
   If $(E_{lion} < 0)$
   then the lion is suspected of being inconsistent, so suppress it:
     $Q_I(lion_{t-1,t-1}) \leftarrow 0.0$
   Else update it using the standard 1-step Q-learning rule:
     $Q_I(lion_{t-1}, oact_{t-1}) \leftarrow Q_I(lion_{t-1}, oact_{t-1}) + \alpha E_{lion}$.
3) Update non-lion internal states:
   For each $s \in S_{t-1}$ and $s \neq lion_{t-1}$ do:
     Let $E_s = r_{t-1} + \gamma V_E(s_t) - Q_I(s, oact_{t-1})$
     If $(E_s < 0)$
     then $s$ is suspected of being inconsistent, so suppress it: $Q_I(s, oact_{t-1}) \leftarrow 0.0$
     Else update it using the lion's error:
       $Q_I(s, oact_{t-1}) \leftarrow Q_I(s, oact_{t-1}) + \alpha' E_{lion}$ — where $\alpha' < \alpha$.
4) Also use inconsistencies in the role-model's behavior to suppress inconsistent decisions:
   For each $(s,a) \in S_{t-1} \times A_O$ do:
     If $(a = a^o_{t-1}$ and $B(s,a) < 0)$ or $(a \neq a^o_{t-1}$ and $B(s,a) > 0)$
     then $(s,a)$ is suspected of being inconsistent, so suppress it: $Q_I(s,a) \leftarrow 0.0$
5) Update the bias function, $B$:
   for all $(s,a) \in S_{t-1} \times A_O$ do: $B(s,a) \leftarrow (1-\psi)B(s,a) + \psi r_o(s,a)$.
```

Figure 7.7: An outline of the decision procedure used by Meliora III-LBW in the watching mode. This procedure generates a set of candidate internal states to represent the current situation facing the role model, observes the role model's action, and updates the action-value function and a bias function based on the results. An internal state is considered inconsistent if the role model is observed to perform more than one action for that state.

representations for the current situation, S_t. From these candidates, a single lion is identified to represent the current state. The perceptual cycle is not shown in the figure since it is identical to the one used by Meliora II. Next, instead of selecting an action to execute, the robot simply observes the action executed by the external role model, a_t^o, and the reward that results, r_t^o.[2] At this point, the robot has made an observation, O_t, which consists of the following information: S_t — a set of candidate internal states; a_t^o — the overt action command executed by the role model; and r_t^o — the reward received by the role model as a result of its action. This information is used to update the action-value and bias functions as follows.

The bias function is updated by increasing the bias of state-action pairs that match the role model's behavior and by decreasing the bias of those decisions that do not. At time t, the bias function is updated as follows:

```
For all $(s,a) \in S_{t-1} \times A_O$ do:
  $B(s,a) \leftarrow (1-\psi)B(s,a) + \psi r_o(s,a)$,
```

where,

$$r_o(s,a) = \begin{cases} +R_o & \text{if } a = a_{t-1}^o \\ -R_o & \text{otherwise} \end{cases}$$

and where R_o is a positive constant and ψ is a fixed learning rate parameter between 0 and 1.

The overt action-value function is updated using the same technique as in the lion algorithm. That is, the lion is updated using the 1-step Q-learning rule and non-lions are updated based on the error in the lion's estimate. As before, decisions that are suspected of being inconsistent have their action-values suppressed. In Meliora-III-LBW, inconsistent decisions are detected by using the overestimation technique and by detecting variations in the role model's choice of action for a given internal state. In particular, if for a given internal state, the role model is observed to perform two different actions, then the internal state is assumed to be inconsistent.[3] The bias function is used to detect variations within internal states as follows:

```
For all $(s,a) \in S_t \times A_O$ do:
  the decision $(s,a)$ is suspected of being inconsistent and suppressed if:
    1) $a = a_t^o$ and $B(s,a) > 0$
  or
```

[2]The details of how this recognition is performed are non-trivial and may be very difficult. However, in our experiments these difficulties are ignored.

[3]This method for detecting inconsistent states assumes that the role model always performs the same optimal action for a given state, even when more than one exists.

2) $a \neq a_i^o$ and $B(s,a) < 0$.

Both of these cases correspond to situations where the learner has previously observed the role model perform a different action in a situation represented by s.

7.3.2 Experiments

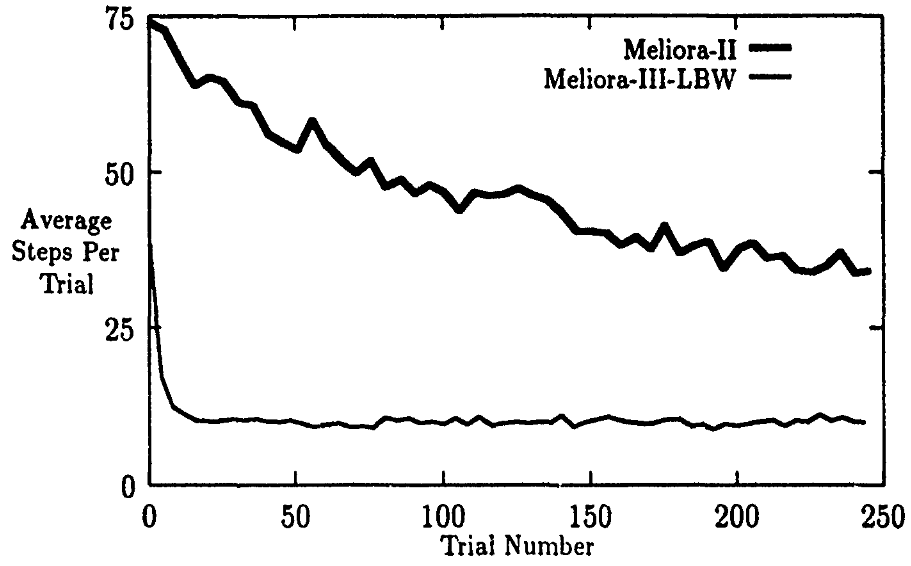
To evaluate the utility of LBW for state identification and overt control, Meliora-III-LBW was tested on the GB-task. In these experiments the robot alternates between solving the task itself and watching an external role model perform the task. As before, experimental runs consist of 250 trials each. In these experiments the following parameter values were used: $\alpha = 0.2$, $\gamma = 0.8$, $\psi = 0.6$, $p = 0.9$, $p' = 0.9$, $R_o = 500$, and $n_{quit} = 100$. Results are shown in Figure 7.8, which shows the average number of steps per trial and the average suppression rate per trial for Meliora-III-LBW and Meliora II. The average solution time curve for Meliora-III-LBW shows only trials attempted by the robot. Both plots clearly indicate that experiences gained by observing a skilled role model substantially improve overall performance.

7.4 Analysis

This section presents a more formal analysis of search in the unbiased Q-learning, LEC, and LBW algorithms. Naturally the scaling properties of any reinforcement learning algorithm strongly depend upon the structure of the decision problem and the details of the algorithm itself. To date, we have been unable to analyze the learning time complexity of Q-learning, LEC, or LBW in general. However, results have been obtained for specific algorithms on a restricted class of deterministic decision problems. In particular, it is shown that for a rather restricted (but representative) class of generic decision problems, called ***homogeneous problem solving tasks***, the learning time of a zero-initialized Q-learning system scales at least exponentially in the depth of the state space.[4] LEC and LBW algorithms are shown to have substantially better learning time complexities. In particular, an LEC algorithm is shown to have an expected learning time that is no worse than linear in the state space size, and under appropriate conditions linear in the length of the optimal solution path (and independent of the state space size). Analogous results are obtained for LBW algorithms.

[4]A zero-initialized Q-learning system is a system whose initial action-values are uniformly zero. Also, as will be defined below, the depth of a state space corresponds roughly to the maximum number of steps between two states in a state space.

a)



b)

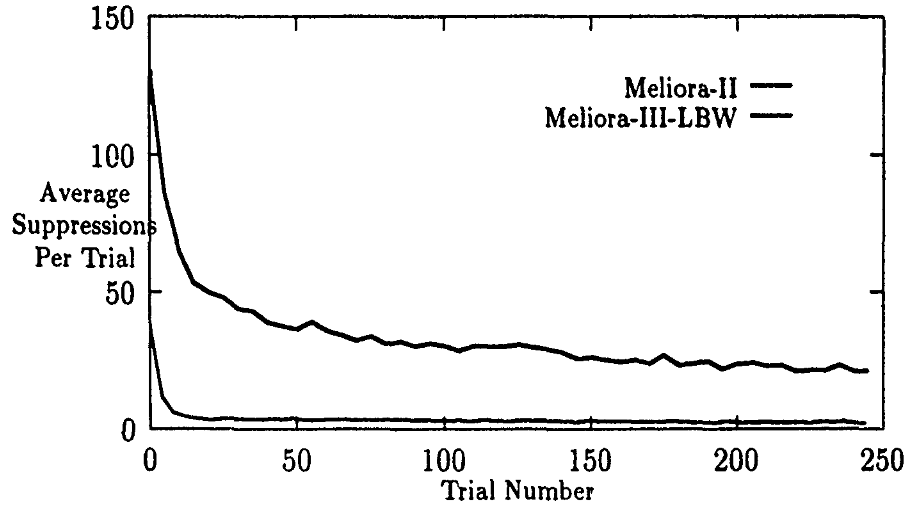


Figure 7.8: Plots of a) the average solution time and b) the average number of suppressions per trial for Meliora-III-LBW and Meliora-II.

The analysis begins by defining a number of properties that are useful for characterizing state spaces. These properties are then used to define the class of homogeneous decision problems. Next, a lower bound on the expected learning time for a zero-initialized Q-learning system is derived, and its relevance to Q-learning in general is discussed. Following that, results for LEC and LBW algorithms are presented and discussed. For the most part, the proofs for the theorems that follow are straightforward. They have been omitted for clarity of presentation, but can be found in Appendix A.

7.4.1 Definitions

Let us begin by defining a number of properties that are needed to define the class of homogeneous problem solving tasks.

Definition 1 (deterministic decision problem) *A decision problem is* deterministic *if it can be described by a Markov decision process whose transition and reward functions are true functions (i.e., deterministic).*

Definition 2 (1-step invertible) *A deterministic decision problem is* 1-step invertible *if every action has an inverse. That is, if in state x, action a causes the system to enter state y, there exists an action a^{-1} that when executed in state y causes the system to enter state x.*

Definition 3 (uniformly k-bounded) *A state space is* uniformly k-bounded *with respect to the state x if*

1. *The maximum number of steps needed to reach x from anywhere in the state space is k.*
2. *All states whose distance to x is less than k have b_- actions that decrease the distance to x by one, b_+ actions that increase the distance to x by one, and $b_=$ actions that leave the distance to x unchanged.*
3. *all states whose distance to x is k have b_- actions that decrease the distance by one and $b_= + b_+$ actions that leave the distance unchanged.*[5]

Definition 4 (homogeneous) *A state space is* homogeneous *with respect to the state x if it is* 1-step invertible *and* uniformly k-bounded *with respect to x. In this case, k said to be the* depth *of the state space.*

[5]That is, at the boundaries, actions that would normally increase the distance to x are folded into actions the leave the distance unchanged.

Definition 5 (polynomial width) *A homogeneous state space (of depth k) has* polynomial width *if the size of the state space is a polynomial function of its depth.*

Definition 6 (problem solving task) *A* problem solving task *is a sequential decision problem in which*

1. *each trial begins in a designated start state S,*

2. *each trial ends when the system reaches a designated goal state G, or gives up, and*

3. *the system receives a non-zero, positive reward only upon entering the goal state. That is,*

$$r_t = \begin{cases} 1 & \text{if } x_{t+1} = G \\ 0 & \text{otherwise} \end{cases} \tag{7.5}$$

Definition 7 (homogeneous problem solving task) *A task is a* homogeneous problem solving task *if it is a problem solving task and its associated state space is homogeneous with respect to the goal state G.*

Homogeneous problem solving tasks represent an idealization of decision problems commonly studied in reinforcement learning. For instance, properties such as locality and invertibility of actions, and delayed, sparse rewards are common to many sequential decision problems. Other assumptions, such as the uniformity of the state space, the deterministic effects of actions, and the use of single initial and final states are somewhat restrictive; however, they simplify the analysis. Some of these restrictions are not strictly necessary, but simplify exposition. For instance, many of the results described below also apply to stochastic processes, so long as transitions are local. Other assumptions, such as the uniformity of the state space, though unrealistic, probably do not fundamentally affect complexity of a given algorithm, but are needed to obtain closed form analytical expressions. For instance, the scaling properties of an algorithm when applied to non-uniform tasks are likely to match the scaling properties of the algorithm when applied to an analogous class of uniform tasks. Also, uniformity of the state space provides a convenient means for clearly defining what it means to scale a task, and allows us to carefully model and evaluate the effect of structural bias in the state space (e.g., by manipulating the values for b_+, b_-, and $b_=$).

7.4.2 Unbiased Random Walks and Q-learning

In general, it would be desirable to have bounds on the expected time needed to solve a problem initially. Such bounds would be useful since much of the time spent learning is accounted for in the first few trials when the agent thrashes about in search of feedback (reward). Unfortunately, even for homogeneous tasks, this search time cannot be determined without knowledge of the learner's initial parameter values. For instance, proper initialization of a Q-learner's action-value function will yield optimal performance from the outset. If the initial parameter values do not encode any information about the task, then the learner is said to be *initially unbiased*. In Q-learning, initially unbiased agents can be obtained by using constant initial action-values (i.e., $\forall_{(s,a)\in S\times A} Q(s,a) = C$). Even under these circumstances the initial performance of the learner is difficult to quantify analytically. A special case that is analytically tractable occurs when the initial action-value function is uniformly zero.

Definition 8 (zero-initialized) *A Q-learning system with an action-value function whose initial values are uniformly zero is said to be* zero-initialized.

In the case of zero-initialized Q-learning, the agent performs an unbiased random walk over the state space until it first encounters a non-zero reward. Here we assume standard semantics for Q-learning, namely that the agent when following policy selects the action with the largest action-value and breaks ties by randomly selecting one of the maximal actions. In a zero-initialized system, all actions initially appear equally good (or bad) since they all share the same action-value. Moreover, the estimation error, obtained after each step and used to update the action-value function, is zero until the agent receives a non-zero reward. Thus, in a problem solving task, a zero-initialized Q-learner solves its first task by performing a random walk.[6] For homogeneous state spaces, a closed form expression can be obtained for the expected duration of this random walk.

Theorem 3 *In a homogeneous problem solving task, the expected time needed by a* zero-initialized Q-learning *system to perform the random walk needed to solve the first trial is given by the expression*

$$c_1 * \left\{ c_2 \left[\frac{P_+}{P_-}\right]^{k-i} \left[\left(\frac{P_+}{P_-}\right)^i - 1\right] + \frac{i}{1-2P_+} \right\} \tag{7.6}$$

where

$$c_1 = \left(\frac{1}{1-P_=}\right), \quad c_2 = \frac{P_+}{(1-2P_+)^2},$$

$$P_+ = \frac{b_+}{b_+ + b_-}, \quad P_- = 1 - P_+,$$

[6]Indeed, since the reward received at the end of the task may only be used to update a few states (exactly 1 in the case of 1-step Q-learning), the agent will perform a series of shorter and shorter random walks, one for each trial until reward information gets propagated to more distal states.

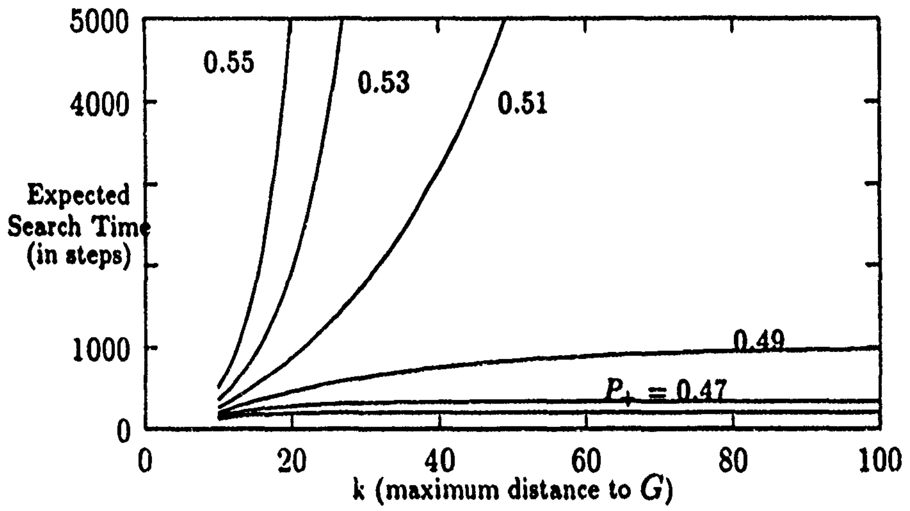


Figure 7.9: Search time complexity as a function of state space depth.

and

$$P_= = \frac{b_=}{b_= + b_+ + b_-},$$

and where i is the distance between S and G, and k is the depth of the state space (with respect to G).

Corollary 4 *For state spaces of fixed width and for $P_+ > 1/2$, the expected search time is exponential in the state space size.*

Corollary 5 *For state spaces of polynomial width and for $P_+ > 1/2$, the expected search time is moderately exponential in the state space size.*

In Theorem 3, $P_=$ is the probability that the system, when choosing actions randomly, selects an action that leaves the distance to the goal unchanged, and P_+ (and P_-) is the conditional probability that the system chooses an action that increases (decreases) the distance to the goal, given that it chooses one that changes the distance. These transition probabilities capture the structural bias inherent in the underlying state space. That is, for some problems the inherent structure of the task is such that it tends to funnel the agent toward the goal. In other cases, the structure of the state space may negatively bias the random walk and substantially increase its expected duration.

Figure 7.9 shows a series of plots of expected solution time (Equation 7.6) versus state space depth k for $i = 10$, and $P_+ \in [0.45, 0.55]$. When $P_+ > 1/2$, the expected solution time scales exponentially in k, where the base of the exponent is the ratio $\frac{P_+}{P_-}$. When $P_+ = 1/2$, the solution time scales linearly in k, and when $P_+ < 1/2$ it scales sublinearly.

The case where $P_+ > 1/2$ (negative biasing) is important for two reasons. First, for many interesting problems it is likely that $P_+ > 1/2$. For exam[illegible]e, if a robot attempts to build an engine by randomly fitting parts together, it is much more likely to take actions that are useless or move the system further from the goal than towards it. This follows since engine assembly requires a fairly sequential ordering. Similarly, a child can be expected to take time exponential in the number of available building blocks to build a specific object when combining them at random. Of course, the state spaces for building engines and assembling blocks are not homogeneous, but the negative bias inherent in their state spaces are likely to have similar exponential effects on agents that initially solve tasks using random walks.

Second, when P_+ is only slightly greater than 1/2, it doesn't take long before the exponent leads to unacceptably long searches. Figure 7.9 illustrates this point dramatically; even when P_+ is as small as 0.51 the solution time diverges quickly. When $P_+ = 0.55$ (i.e., the system is only 10% more likely to take a "bad" action than a "good" one), the search time diverges almost immediately.

Theorem 3 applies only to zero-initialized Q-learning systems on homogeneous problem solving tasks. When the task is non-homogeneous, the analysis breaks down because the expected time needed to perform the random walk is difficult to analyze. When the initial action-values are non-zero the analysis breaks down because the agent's behavior is no longer completely random. For instance, initializing the action-values to a fixed positive constant yields a search that is biased towards exploring previously untried actions. This follows since in this case each time the action-value of a state-action pair is updated its value is reduced. The more a state-action pair is executed the lower its action-value becomes. This, in turn, favors the selection of actions that have been tried less often in the past, resulting in more exploration than in a pure random walk. Conversely, initializing the action-values to a fixed negative constant yields a search that avoids exploration and prefers to repeatedly try previously used actions. This follows since in this case each time a state-action pair is applied its action-value is increased incrementally toward zero. The more a decision is selected, the greater its action-value becomes and the more likely it is to be selected in the future.

Theorem 3 states that for $P_+ > 1/2$, a zero-initialized Q-learner can be expected to take time exponential in the depth of the state space for the initial solution in a homogeneous problem solving task. It is useful to determine if similar complexity results hold for other initial action-values. Since these cases are difficult to analyze formally, this question is addressed empirically. In particular, three separate 1-step Q-learning agents were applied to a homogeneous problem solving task whose state space was systematically scaled in depth. The three agents, named Q-, Q0, and Q+, had action-value functions that were uniformly initialized to -1.0, 0.0, and 1.0, respectively. The decision task is shown in Figure 7.10. The goal state, G, is on the far left. The start state, S, is maintained at a constant distance of 5 steps from the goal, as the state space is scaled in depth, k. Because there is a 3:1 ratio of actions that increase the distance to the goal to actions that decrease the distance to the goal, the state space is biased against that goal state. The parameters that characterize the state space in terms of Theorem 3 are: $b_+ = 3$, $b_- = 1$, $b_= = 2$. The depth of the state space was systematically scaled from $k = 5$ to $k = 100$. For each depth, the expected number of steps needed to solve the first trial was estimated by averaging the results of 200 runs.[7] The average first solution times for Q0 and Q+ are plotted versus depth in Figure 7.11a. Q- was unable to solve the task in a reasonable amount of time under any circumstances since it tended to get stuck in local cycles that never made progress toward the goal. The figure shows that Q+, aided by its exploratory bias, significantly outperforms Q0. However, it continues to scale super-linearly in the depth of the state space. To determine whether the search time scales exponentially in depth, the logarithm of the average solution time is plotted versus depth in Figure 7.11b. This plot shows that, as expected, Q0 scales exponentially in depth, but that Q+ scales sub-exponentially. Apparently, the slight exploratory bias caused by the use of positive initial action-values is sufficient to reduce the complexity of the initial search, even for problem spaces that are intrinsically biased away from the goal. Nevertheless, for practical purposes, the search time continues to lead to intractably long learning times since it appears to be at best polynomial in the depth of the state space.

7.4.3 Analysis of LEC

The principal idea of both LEC and LBW is to reduce initial search by exploiting information gained from external agents. LEC algorithms depend on immediate feedback from a knowledgeable external critic to reduce feedback latency, while LBW algorithms depend on other agents for alternative (often highly biased) sources of experience.

The results presented in this subsection focus on LEC algorithms. In particular it is shown that in a homogeneous state space the BB-LEC algorithm has an expected initial search time that is at most linear in the size of the state space. This upper bound is an improvement over unbiased Q-learning, but still disappointing because of its dependence on state space size. However, tighter upper bounds can be obtained either by restricting the class of state spaces further or by modifying the capabilities of the learner. Under these circumstances bounds

[7]After each run the agent's action-value function was reset to its initial value.

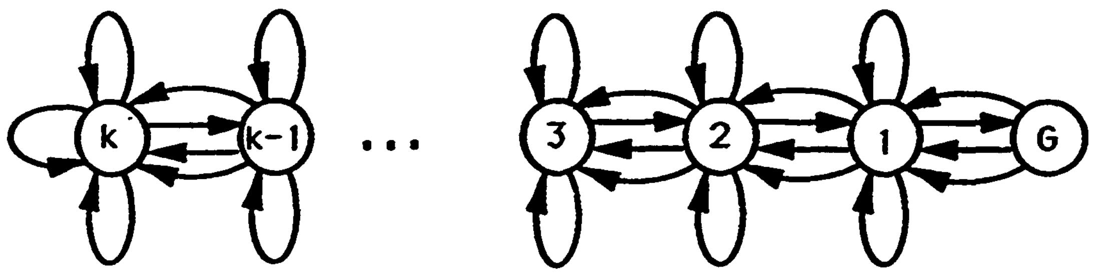


Figure 7.10: The 1-dimensional homogeneous problem solving task used to assess the effect of the initial action-value function on initial search. The start state is maintained at a constant distance from the goal, G, as the depth of the state space is scaled from $k = 5$ to $k = 100$.

on the expected search time can be obtained that 1) depend only on the length of the optimal solution path and 2) are independent of the state space size.

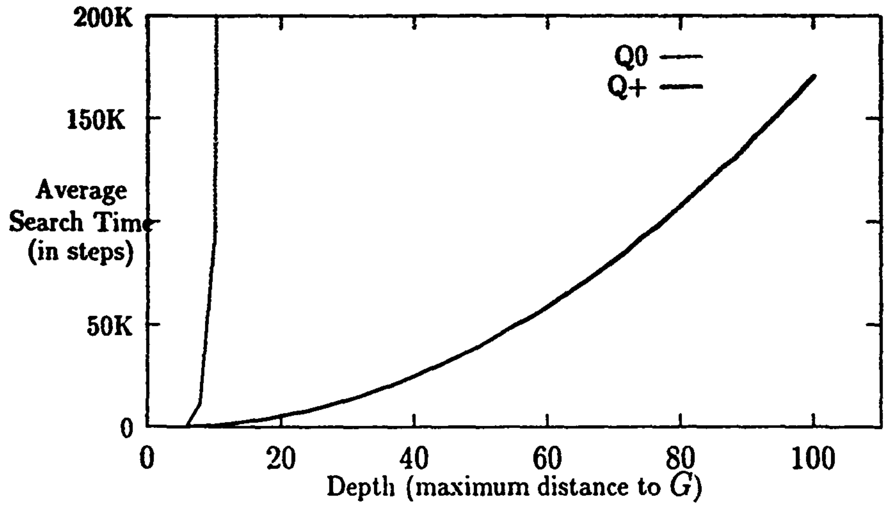
Theorem 6 *The expected time needed by a zero-initialized BB-LEC system to learn the actions along an optimal path for a homogeneous problem solving task of depth k is bounded above by*

$$\frac{k}{P_{\text{critic}}} * |S| * b \tag{7.7}$$

where P_{critic} is the probability that on a given step the external critic provides feedback, $|S|$ is the total number of states in the state space, and b is the branching factor (or total number of possible actions per state).

This upper bound is somewhat disappointing because it is expressed in terms of the state space size, $|S|$, and the maximum depth, k. Our goal is to find algorithms that depend only upon task difficulty (i.e., length of optimal solution) and are independent of state space size and depth. Nevertheless, the result is interesting for two reasons. First, it shows that when $\frac{P_+}{P_-} > 1/2$, BB-LEC is 'n improvement over unbiased Q-learning since the expected search time grows at most linearly in k, whereas)-learning grows exponentially for zero-initialized systems and apparently polyno ..ally for systems with fixed positive initial action-values. Second. because this upper bound is inversely proportional to P_{critic}, the

a)



b)

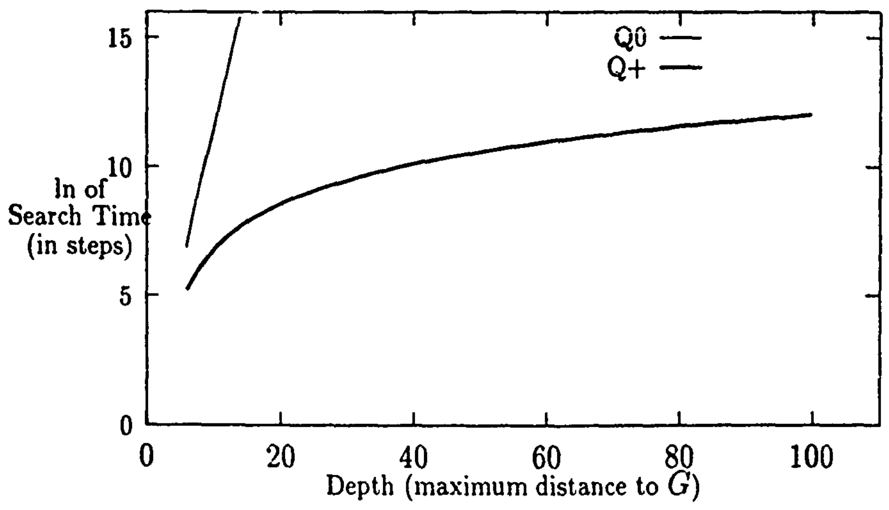


Figure 7.11: Average solution time plots for Q0 and Q+. a) The average number of steps taken in the initial solution to the task for Q0 and Q+ versus depth. The exploratory bias afforded Q+ by its initial positive action-values leads to performance that is significantly better than Q0. b) The natural logarithm of the average first solution time versus depth for Q0 and Q+. The graph confirms the exponential scaling rate for Q0 and shows Q+ to scale subexponentially in depth.

theorem shows that even infrequent feedback from the critic is sufficient to achieve the linear upper bound. This effect was observed in Meliora-III-LEC, where even infrequent feedback from the critic substantially improved performance.

The trouble with the BB-LEC algorithm, as described so far, is that the critic's feedback arrives late. That is, by the time the learner receives the critic's evaluation it finds itself in another (neighboring) state, where the feedback is of no value. If the learner has an efficient means of returning to previously encountered states, it can make better use of the critic's feedback. This idea leads to the following results, which show that under appropriate conditions the search time depends only upon the solution length and is independent of state space size.

Theorem 7 *If a zero-initialized BB-LEC system uses an inverse model* [8] *to "undo" non-optimal actions (as detected based on feedback from the external critic) then the expected time needed to learn the actions along an optimal path for a homogeneous problem solving task is linear in the solution length i, independent of state space size, and bounded above by the expression*

$$\left[\frac{2}{P_{-}(1-P_{=})}-1\right] * i. \tag{7.8}$$

Similarly, if the task is structured so that the system can give up on a trial that it fails to solve after a reasonable amount of time or if the system is continually presented with opportunities to solve new instances of a problem, then previously encountered situations can be revisited without much delay and the search time can be reduced.

Theorem 8 *A zero-initialized BB-LEC system that aborts a trial and starts anew if it fails to solve the task after n_q ($n_q \geq i$) steps has, for a homogeneous problem solving task, an expected initial solution time that is linear in i, independent of state space size, and bounded above by the expression*

$$\left(\frac{1}{P_{-}(1-P_{=})}\right) * n_q i. \tag{7.9}$$

Corollary 9 *A zero-initialized Q-learning system using BB-LEC that quits a trial and starts anew upon receiving negative feedback from the external critic has an expected solution time that is bounded from above by the expression*

$$\left(\frac{1}{P_{+}(1-P_{=})}\right) * \left(\frac{1}{P_{=}+(1-P_{=})P_{+}}\right) * i. \tag{7.10}$$

[8] This theorem does not account for the time needed to learn the inverse model. It assumes the inverse model is known *a priori*.

The crucial assumption underlying these results is that the learner has some mechanism for quickly returning to the site of feedback; however, for some tasks an explicit mechanism may not be necessary to decouple search time from the state space size. In particular, if the optimal decision surface is smooth (i.e., optimal actions for neighboring states are similar), then action-value and bias functions implemented using approximation techniques that locally interpolate (e.g., CMACs, or Neural Nets) can immediately use the critic's feedback to bias the impending decision. Or alternatively, if Q and B are approximated using techniques that generalize non-locally (e.g., classifier systems [Holland *et al.*, 1986]), then the critic's feedback can be expected to transfer to other non-local situations as well. Although not reflected in the above theorems, generalization techniques like these probably will enable LEC to be useful even when explicit mechanisms for inversion are not available.

7.4.4 Analysis of LBW

LEC algorithms are sensitive to naive critics. That is, if the critic provides poor feedback, the learner will bias its policy incorrectly. This limits the use of LEC algorithms to cases in which the external critic is skilled and attentive. *Learning By Watching*, on the other hand, does not necessarily rely on a skilled, attentive critic. Instead, the learner gains additional experience by interpreting the behavior of others. If the observed behavior is skilled so much the better, but an LBW system can learn from naive behavior too.

Theorem 10 *For a population of naive (zero-initialized) Q-learning agents using LBW, the expected time to learn the actions along an optimal path decreases to the minimum required learning time at a rate that is $\Omega(1/n)$, where n is the size of the population.*

Without help by other means, a population of naive LBW agents may still require time exponential in the state space depth. However, search time can be decoupled from state space size by adding a knowledgeable role model.

Theorem 11 *If a naive agent using LBW and a skilled (optimal) role model solve identical tasks in parallel and if the naive agent quits its current task after failing to solve it in n_q steps, then an upper bound on the time needed by the naive agent to first solve the task (and learn the actions along the optimal path) is given by*

$$\left\lceil \frac{i^2}{n_q} \right\rceil n_q + i. \tag{7.11}$$

As with the LEC results, Theorem 11 relies on the agent having a mechanism for returning to previously encountered states. Intuitively this result follows since when a naive agent and a skilled agent perform similar tasks in parallel, it is possible for the naive agent to move off the optimal solution path and find itself in parts of the state space that are never visited by the skilled agent. Starting over is a means for efficiently returning to the optimal solution path. Again, we expect LBW systems to perform well on tasks that have decision surfaces that are smooth or that otherwise lend themselves to function approximation techniques that generalize.

8 Limitations and Future Work

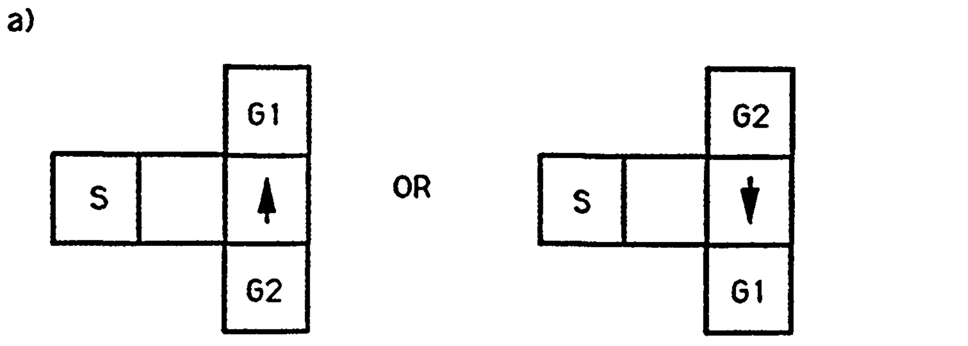
This chapter discusses some of the limitations of the algorithms described so far and points out a number of difficulties that may arise when applying these techniques to more complex and realistic control problems. To overcome these difficulties extensions to the previously described algorithms are proposed. While these extensions seem plausible, they are necessarily speculative. To date none has been implemented or tested. Approaches that fall outside of the CR-method framework are also considered.

8.1 Separation of Perceptual and Overt Control

The fundamental assumption of the CR-method is that at each point in time control is achieved be first identifying the external state (state identification) and then executing the appropriate overt action (overt control). To ensure that the external state does not change, only perceptual actions are allowed to be executed during state identification. This restriction has two important consequences: 1) it constrains the control strategy learned by the agent to one that separates perceptual and overt control control, which may be suboptimal; 2) it limits the class of decision problems that can be solved at all using the CR-method.

8.1.1 Non-Optimal Control

The CR-method restricts the embedded decision system's control strategy to one that performs overt actions only from consistent internal states. While this approach may be able to optimize the expected return v .n respect to this restricted class of control strategies (i.e., strategies that first identify and then control the external state), such systems cannot learn optimal control policies for internal decision problems that require the execution of overt actions from inconsistent



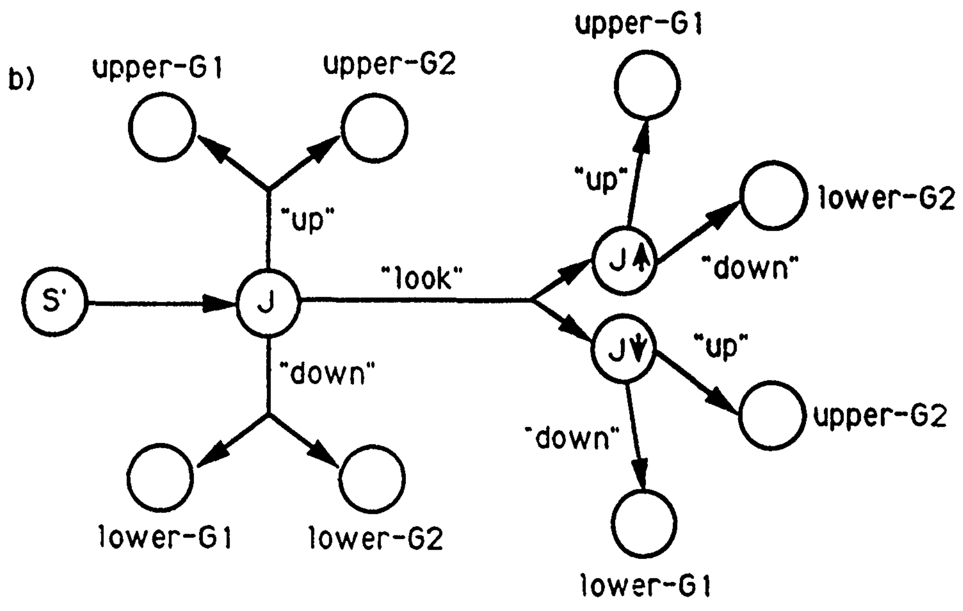


Figure 8.1: A task that CR-methods cannot learn to perform optimally. a) the external task: on half the trials G1 is in the upper cell and G2 is in the lower: on the other trials the rewards are reversed. An arrow on the ceiling of the junction cell indicates the direction to G1, the larger reward. b) the internal decision problem; the inconsistency in state J can be resolved by performing the "look" perceptual action. However, depending on the difference between G1 and G2 and the cost of the "look" action, the optimal action in state J may be to perform the "up" overt action directly.

internal states. An example of such a decision problem is illustrated in Figure 8.1. The figure shows a task in which a control policy that performs an overt action from an inconsistent internal state outperforms the best policy that executes only perceptual actions from inconsistent internal states. The external part of the task involves navigating from a start state to one of two possible reward sites (Figure 8.1a). Trials begin in state S and end when the agent receives a non-zero reward (either G1 or G2). On half of the trials, the agent receives reward G1 upon entering the upper cell, and G2 upon entering the lower cell. On the other half of the trials, the rewards are reversed. At the junction cell there is an arrow on the ceiling, which indicates the direction to G1, the larger reward. This information is not automatically registered by the agent but can be obtained by performing the perceptual action "look" at the junction. The corresponding internal decision problem is shown in Figure 8.1b. The internal state J is inconsistent. It represents the situation where the agent is at the junction in the external task but is not looking at the arrow. In this state, the overt actions "up" or "down" have inconsistent effects that depend upon the direction of the arrow. By performing the "look" action, the agent can resolve the ambiguity, reach a consistent internal state (either $J\uparrow$ or $J\downarrow$), and then perform an overt action that maximizes the expected return. Depending on the difference between G1 and G2 and the cost of executing the perceptual action "look," a policy that executes the overt "up" action in state J may on average outperform policies that always resolve the inconsistency. For instance, if the difference between G1 and G2 is less than the cost of the "look" action, it is better on average to perform an overt action directly from state J. Unfortunately, systems based on the CR-method are constrained to control strategies that always resolve inconsistencies, even if the cost of resolution outweighs the benefit. As long as the decision system is constrained to execute overt actions only from consistent internal states, there will be tasks like this one that the agent cannot perform optimally. The CR-method does not provide a mechanism for trading off the cost of perception against the benefit of acting without knowledge.

The Q-CUP Algorithm

One approach to overcoming this limitation is to abandon the distinction between perceptual and overt actions and to take a more careful look at the failure of Q-learning (*cf.* Chapter 5). Recall that in Chapter 5, it was shown that inconsistent decisions interfere with Q-learning by having action-values that average the expected returns of the external decisions they represent. These inaccurate estimates (utility aberrations), in addition to inaccurately estimating the true utility of performing a given action at a specific point in time, also interfere with estimating the action-values. While it is unlikely that the utility aberrations for inconsistent decisions can be eliminated, it may be possible to prevent them from interfering with the action-value estimates of consistent decisions. That is, suppose we are able to detect inconsistent decisions (say, using the techniques described in the previous chapters). Suppose further that standard Q-learning is used to learn a control policy for both overt and perceptual actions, except that instead of using a fixed updating rule, like the 1-step corrected truncated return, the updating rule is always based on multi-step estimators whose correction term is the utility of a consistent state. That is, after executing a given decision, rewards are collected until the agent encounters a consistent state; at that time a return estimate for updating the action-value for the earlier decision is constructed from the accumulated, appropriately discounted rewards and the utility estimate for the current consistent state. Under these circumstances, it should be possible to estimate accurately the sampled average of the action-values of the external decisions represented. The action-value for an inconsistent decision depends on the relative frequency that external decisions represented by that decision are sampled (or encountered), and in general the inconsistent decision will not accurately estimate the action-values of the external decisions it represents. However, the action-values for consistent decisions should be accurate with respect to the external decisions they represent. We call this integrated approach to control the Q-CUP algorithm (for Consistent UPdating). The Q-CUP algorithm may be able to outperform the CR-method on certain decision problems. For instance, the Q-CUP algorithms should be able to learn the optimal policy for the task in Figure 8.1, since in this case, the sampled average of the return obtained after executing either the "up" or "down" overt actions from state J is greater than the expected return obtained by first looking at the arrow to disambiguate the situation. Also, notice that if the difference between G1 and G2 is large enough to make the "look" action worthwhile, the Q-CUP algorithm should learn to perform that action. While the Q-CUP algorithm appears to be able to outperform the CR-method on the task in Figure 8.1, it may not always be appropriate since the policy it learns may be unstable and its performance may be erratic. This can be demonstrated by modifying the task in Figure 8.1 as follows:

1. Increase the difference between G1 and G2 to some large value (e.g., $G1 = 10$, $G2 = 0$),

2. Adjust the distribution of trials so that G1 is almost always found in the upper cell (e.g., 99% of the time).

3. Change the dynamics of the problem so that an incorrect up or down action returns the system to the start state S.

Under these circumstances, an agent using the Q-CUP algorithm will almost always initially learn to execute the "up" action from state J since with high probability this action will be optimal for the initial series of trials. However, at some point the agent will encounter a trial where the reward is in the lower cell. On this trial, the agent will get stuck in a loop, continually trying the "up" action until the action-value for the "up" action in state J is reduced enough to permit another action or until the agent selects an alternate action at random. This oscillation in the agent's action-value function will tend to repeat indefinitely and its performance will continue to be unnecessarily long for the uncommon case in which the reward is in the lower cell.

On the other hand, an agent using the CR-method on this task would have an action-value function that is more stable. While it might be slightly less efficient in the common case (reward up) it would be significantly more effective in the uncommon case (reward down). Indeed, depending upon the learning rate and exploration strategy used, an agent using the CR-method may on average outperform an agent using Q-CUP.

The Q-CUP algorithm has not yet been implemented or tested, and it is difficult to speculate on its specific performance characteristics. However, because it does not distinguish between perceptual and overt actions, it overcomes one of the major limitations of the CR-method. Whether it works well in practice or has limitations of its own remains to be seen.

Neural Network Algorithms

Other alternatives to the CR-method that allow for the free intermixing of overt and perceptual actions are some of the recent neural network algorithms [Jordan and Rumelhart, 1990; Schmidhuber, 1990b; Thrun and Moller, 1991; Nguyen and Widrow, 1989]. In these algorithms, no explicit estimate of the future expected return is maintained or used to update the controller's policy. Instead, adaptation of the mapping between sensory inputs and control actions is achieved by modifying the weights of the network using a gradient descent procedure. In particular, at each point in time, the gradient of the immediately received reward is computed with respect to the weights in the network (using backpropagation). These immediate gradients, accumulated over time, are then combined to yield an approximation of the gradient for the total discounted return, which in turn is used to adjust the network's performance. Since these methods do not depend upon explicit utility estimates, as in Q-learning, they do not suffer as much from utility aberrations. However, they do have a number of drawbacks of their own. First, since the network is adjusted based on actual rewards received (and has no notion of an underlying Markov decision process), these algorithms may suffer from the same instabilities expected to plague the Q-CUP algorithm. Second, in order to perform credit assignment properly for temporally delayed rewards, it is necessary to perform backpropagation steps that go back in time. This process requires the system to keep a history of neuron activations and weight values, and requires a differentiable model of the domain's forward dynamics (used during

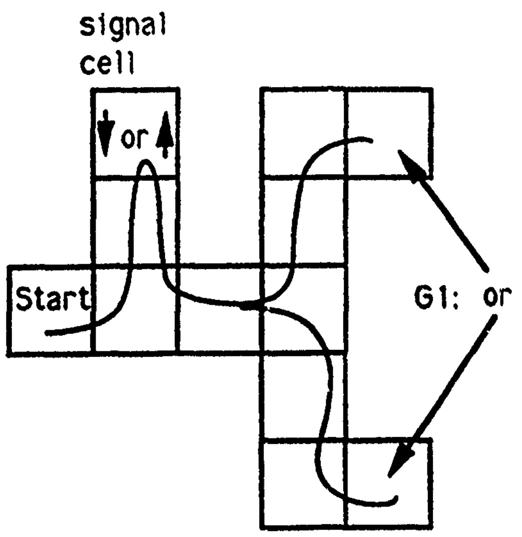


Figure 8.2: A navigation task that requires memory. In this task the position of the reward is indicated by the arrow in a signal tile. We assume the agent can only sense features of the cell it currently occupies. To solve this task we would like the agent to first determine and remember the direction of the arrow and then navigate to the appropriate location. The systems described in this dissertation have no means of remembering and using any information other than that which is immediately perceivable.

backpropagation). Finally, since these algorithms are based on backpropagation (an approximate gradient descent method), they may get stuck in local minima and fail to converge on an optimal policy. Jordan [Jordan and Rumelhart, 1990] and Schmidhuber [Schmidhuber, 1990b] have done the most work on these algorithms. Schmidhuber, in particular, has described an algorithm for recurrent networks that is capable of learning decision tasks that are non-Markov. However, the results reported so far are preliminary and involve very simple problems (quite a bit easier than the GB-task). In their current state, these algorithms do not appear to be scalable to more complex tasks. Nevertheless, additional research may yield more powerful algorithms based on these direct gradient methods.

8.1.2 Restrictions on the Task Domain

A second limitation of the algorithms described so far is that they are applicable only to tasks in which every external state can be identified using perceptual actions only. This restriction excludes tasks in which the agent must perform overt actions to gain needed state information. For example, in the block stacking domain, tasks that require the agent to move blocks in order to identify the external state (e.g., unstack a block in order to see what is behind it) are not permissible. Navigation tasks where the robot cannot discern the external state from immediately visible cues are also beyond the scope of the present algorithms. An example of this type is shown in Figure 8.2. In this task, the robot's objective is to navigate to a cell that contains food (a reward). As in the task depicted in Figure 8.1, the position of the reward varies from trial to trial and is indicated with an arrow that is located in one of the cells. If the robot's internal state at each point in time is only determined by attributes of the cell it immediately occupies, then a robot will be unable to learn a reliable control strategy since it will be unable to remember the direction of the arrow in the signal cell. Ideally, the agent would execute overt actions that navigate to the signal, sense and remember the value of the arrow, navigate to the junction, recall the information about the arrow, and select the next appropriate move. Unfortunately, the systems described in this dissertation support neither the use of overt actions for state identification (navigating to the signal cell) nor any mechanism for recording and recalling relevant aspects of previously visited positions. Both of these processes are necessary to solve the task in Figure 8.2.

Even though the Q-CUP algorithm allows for the execution of overt actions in inconsistent states, the Q-CUP algorithm will also fail to solve this task without any means for explicitly remembering the value of the signal arrow once the robot leaves the signal cell. A simplifying assumption implicit in all of the algorithms described so far is that the internal state space is defined solely in terms of the agent's immediate precepts. The agent is totally situated in that after each overt action its internal state is essentially flushed and reacquired based only on immediate sensory inputs. Clearly, this is a very restrictive assumption. Unfortunately, very little is currently known about how to extend these state space models efficiently to include features of past situations. This limitation represents an exciting challenge to this approach to adaptive perception and action. Recent work on learning finite state automata by exploration may provide a reasonable starting point for attacking this problem [Rivest and Schapire, 1987; Mozer and Bachrach, 1989]. Approaches that distinguish internal states based on differences in the agent's transition history [McCallum, 1991] and approaches that combine the CR-method with buffered sensory inputs or deictic memory modules may also yield useful algorithms.

Reinforcement learning algorithms based on recurrent neural networks offer another possible approach to this problem since the output of a recurrent network at a given point in time can depend upon the total sequence of inputs it has received up to that time [Schmidhuber, 1990c; Williams and Zipser, 1988; Simard, 1991; Pineda, 1987]. In principle, these networks can learn to encode and remember relevant past information in their hidden units as needed to learn an optimal control policy. As of this time, I have not experimented with adaptive controllers based on recurrent neural networks. However, in light of their tendency to get stuck in local minima, it is doubtful that existing algorithms can be used to solve the task in Figure 8.2. Nevertheless, continued experimentation with recurrent neural networks for adaptive perception and action is certainly warranted.

8.2 State Space Size

Another issue that requires further examination is the rate at which the size of consistent state spaces grows as the complexity of a decision task is increased. For example, when a task can be accomplished in several different ways (any of which might be optimal), the size of the minimal consistent representation for that task tends to grow as the product of the sizes of the state spaces needed to represent each of the individual approaches to the task. An example from the block stacking domain illustrates the problem more clearly.

Recall that for the GB-task, each pile contains exactly one green block. This restriction was imposed to ensure that the robot's two marker sensory-motor system is adequate to register all the information necessary to represent the task consistently. If two or more green blocks are allowed in the pile, then a consistent representation (with respect to picking up a green block) must encode information about the amount of work needed to clear and pick up each individual green block. That is, to solve the task optimally, the robot must analyze each green block to determine which one is easiest to pick up. For the GB-task (with one green block) the attention frame marker is used to register the number of blocks above the green block, and each state in the consistent internal representation is associated with a unique stack height. For a GB-task with multiple blocks, a marker is needed to register the stack height of each green block in the pile, and each consistent internal state represents a unique combination of stack heights — resulting in a state space that is roughly an n-fold product of the state space for the single block GB-task (where n is the number of green blocks in the pile). This stack height information is needed to consistently represent the state of the external world. Failing to mark and register even one green block opens the door for inconsistencies since that unmarked green block may be the easiest one to pick up.[1]

Similar, but even more dramatic growth in the state space size occurs for more complex tasks. For instance, suppose the robot receives a reward for assembling

[1]Strictly speaking, the minimal consistent internal representation need not encode the stack heights of each and every green block. Actually, only knowledge of the easiest green block, its stack height, and some hand information is needed to define the consistent states. However, in order to determine the easiest block to pursue, the agent must compare the stack heights of each of the green blocks, and given the organization of the sensory-motor system this comparison requires marking and registering the stack heights of each block.

configurations of several blocks. Take the stack red-on-green-on-blue, for example. In this case, to perform optimally the robot must consider all possible combinations of red, green, and blue blocks that could be used to construct the desired stack. For optimal control, it is not sufficient to find just any red, green, and blue blocks; all combinations must be considered and information about all red, green, and blue blocks must be registered.

Large state spaces also result when there are multiple sources of reward. A robot that is rewarded for picking up a green block or for assembling the stack red on blue must consider which one of the subtasks to pursue. A rat that receives rewards for satisfying its diverse bodily needs (feeding, drinking, resting, mating, etc.) must in general consider the appropriateness of pursuing each activity in light of the current situation, and consider the effect of pursuing one activity on its ability to satisfy other needs later.

In all of these examples, large state spaces are needed to guarantee that the agent has sufficient information to perform optimally these multifaceted tasks. Unfortunately, these large state spaces 1) put an unrealistic burden on the sensory system (to track and monitor continually every potentially relevant object) and 2) lead to slower learning of each individual subtask by inhibiting passive abstraction/generalization with an overly large state space.

One approach to reducing this scaling problem is to introduce additional structure into the decision system that partitions the overall decision problem into its constituent pieces. That is, instead of using a single monolithic internal representation and a single monolithic policy function, the decision system is composed of a set of modules, each of which is responsible for learning to represent and perform a single subtask of the overall decision problem. We call one of these modules a *schema*. In the case of a block stacking task where multiple configurations yield a reward (e.g., holding a green block or building a red-on-blue stack), each schema would learn to represent and perform consistently one of the subtasks.

Roughly, each schema would correspond to a complete decision system in itself. It would consist of an identification routine and an overt control component; it would generate and maintain its own internal representation; and it would use the same algorithms described previously to adapt itself. The key difference is that each schema's adaptation would be based on a single (restricted) type of reward (i.e., the reward associated with the subtask it was performing). This reward type could be identified either *a priori* (say through an innate set of different reward types such as hunger-reward, thirst-reward, etc.) or by a set of specific learned conditions that classify the reward (e.g., reward consistently associated with holding a green block). In any case, the internal representation learned would be aimed at *consistency with respect to a specific narrow class of reward*, and the control policy would aim to maximize the accumulation of the specific reward only.

Another important aspect of schemas would be the explicit incorporation of variables (or entities) which could be bound to specific objects in the external world and used to instantiate different instances of a schema. For instance, a single schema could be learned to control the process of picking up a single green block. This scheme might have a single variable, which would be bound to the specific green block to be picked up. At any point in time, the utility values of the internal state of this schema would represent the utility of the external world with respect to picking up the green block bound to the schema. Given a single schema of this type, the agent could then perform a series of "bind-and-evaluate" operations to search for the best green block to pursue when more than one was present.

In general, given a set of schemas and their instantiations, overall control could be achieved by first evaluating the state of the external world with respect to each instantiation and then choosing to follow the actions dictated by the schema with the largest utility.

The major disadvantage of this modular approach is that it may lead to non-optimal control. That is, by greedily performing the schema with the largest utility, the agent may fail to maximize its long-term reward since it may be possible that execution of another schema, which itself yields a lower immediate return, sets up a third schema that yields a larger reward. If the cumulative return of these two schemas is greater than the return achievable by the single maximal schema, then non-optimal performance results. For these cases, more global algorithms (like a monolithic decision system) that consider the utilities of performing combinations of tasks would perform better. However, in many cases, a greedy algorithm may suffice.

One of the major advantages of the proposed modular approach is that it reduces the overall size of the internal state space and policy function. For a monolithic decision system the state space (and the domain of the decision policy) has a size that scales as the cross product of the sizes of each individual subtask, whereas the modular approach has a total space requirement that scales linearly in the number of subtasks (i.e., the total size is the sum of the individual state spaces).

Also, the total sensing requirements of the agent can be reduced by committing to the complete execution of a given schema. That is, at the beginning of a task, the agent could evaluate the utility of pursuing each schema and select the best one for execution. If the agent then commits to performing that activity, it no longer needs to perform subsequent sensing operations to evaluate the other schema. In general, any number of high level control strategies can be used to trade off sensing and overt performance.

A third advantage of this modular approach is that the time needed to learn individual schemas should be substantially reduced compared to the learning time for a monolithic decision system. This follows since the individual internal state spaces needed to represent each subtask are significantly smaller than the state space needed to represent consistently the conjoined monolithic task.

To date, we have done very little work on developing this schema-based control architecture other than to recognize the need for it and speculate on how schemas could be learned and controlled. Although many issues remain unresolved, we are confident that a modular approach of this type will be useful for managing the state space size and the learning rate as task complexity is increased.

8.3 Other Areas for Future Research

There are a number of other topics for future research that lie on the critical path between the ideas proposed in this dissertation and their useful application to real systems.

The study of active perception is in its infancy and is an area that needs further development. The simulated sensory-motor system used in Meliora is overly simplistic, unrealistic, and specifically designed to match the needs of the GB-task. The deictic sensory-motor systems used by Agre [Agre, 1988] and Chapman [Chapman, 1990b], though more sophisticated and more general purpose, are also simulated. These principles of active perception need to be validated on real physical systems. Experimentation on real systems, in addition to validating the feasibility of active perception and deictic representations, is also likely to suggest new visual analysis operations and indexing strategies that might have been overlooked otherwise [Swain, 1990; Wixson, 1991].

It is also necessary to develop more efficient implementations for the CR-method. Meliora used tables to implement the various algorithms described above. While algorithms based on tables are convenient for exposition purposes and easy to implement, they require too much space for larger scale tasks. Implementations based on neural networks or other more concise function approximation techniques need to be explored. Also Meliora's internal state space was defined precisely by the input vector generated by the sensory-motor system. When this vector contains redundant or useless information, it introduces unnecessary distinctions between states and increases the size of the internal representation. An approach that combines the perceptual control process used in Meliora and the input generalization techniques described by Chapman and Kaelbling [Chapman and Kaelbling, 1991] or Tan [Tan, 1991b] could lead to even more compact and task-specific internal representations.

Also, if reasonable learning rates are to be achieved, the cooperative mechanisms described in Chapter 7 (or other algorithms like them) must be developed further. Techniques for recognizing and interpreting the behavior of others is central to this endeavor. More powerful communication and signaling protocols can also be expected to facilitate learning.

Finally, perhaps the most important work that can be done to validate and advance the ideas presented here is to apply them to the control of a real physical system — perhaps a real block-stacking robot, or a robot that learns other children's games (e.g., piece-in-hole games). Building such a system would no doubt uncover any number of assumptions and difficulties that have been unknowingly abstracted out of the formal model and our simulated environment. Just a few issues that are likely to arise include: the weaknesses of discrete state-time control models, the need for memory and state, the need for hierarchical structures to organize both the fine grained and large-scale structure of realistic tasks, and the need to perform certain actions simultaneously.

9 Conclusions

Control is the process of directing interaction with an external environment in order to bring about some desirable outcome. This process involves both sensing and action — sensing to gain information and action to effect change. In both classical control theory and AI it is common to begin by assuming that the control system has a fixed set of sensors that provide it with all the relevant information needed for decision making. In AI, it is common to assume the existence of an objective internal representation that uniquely labels and describes properties of all of the potentially relevant objects in a task domain. While this assumption may be reasonable for tasks that are narrow in scope or well understood in advance, it is unrealistic for more complex tasks that have diverse sensory requirements or that are poorly understood ahead of time. Under these circumstances systems that have flexible but limited access to the environment are more appropriate. The Visual Routines model of human spatial vision [Ullman, 1984], subsequent work on deictic representations [Agre and Chapman, 1987; Agre, 1988; Chapman, 1990b], and other recent work [Ballard, 1991; Swain, 1990; Rimey and Brown, 1990; Wixson, 1990; Wixson and Ballard, 1991] demonstrate the potential utility of active perception. This dissertation extends this line of research by considering the application of reinforcement learning to the adaptive control of perception and action. Our aim has been to develop algorithms by which an autonomous agent can learn not only the overt actions needed to perform a task, but also perceptual control strategies for generating an adequate, yet efficient, task-specific internal representation.

To study this problem a series of programs, collectively called Meliora, were developed in an attempt to build a system that could, using active perception and a deictic representation, learn to solve simple block manipulation tasks. Using the block-stacking task as a guide, a formal model of the control problem facing embedded decision systems was described. This model formalizes the effect of an agent's sensory-motor system in establishing the relationship between an abstract Markov model of a task and the decision problem seen by the embedded controller. The model is used to show that intelligent systems invariably face decision problems that are highly non-Markov and that cannot typically be learned using standard reinforcement learning methods. It was shown that improper control of the sensory system leads to perceptual aliasing — states in the internal representation that confound functionally disparate states in the Markov model of the external task. Perceptual aliasing was shown to interfere with reinforcement learning by prohibiting, at certain points in time, the accurate estimation of the utility of performing an action.

A new decision procedure, called the Lion algorithm, was developed to overcome these difficulties and was demonstrated on the GB-task. The Lion algorithm was shown to be a specific instance of a more general technique, called the *Consistent Representation* (CR) method. The principal idea of the CR-method is to separate control into an identification stage followed by an overt control stage. During identification the agent performs perceptual actions to collect information needed to generate a consistent internal state. During overt control, this consistent state is used to guide selection of the next overt action. Both the identification and overt control stages are adaptive. In Meliora, adaptive overt control was achieved using variations on 1-step Q-learning. Adaptation of the identification procedure is based on detecting and eliminating inconsistent internal states from the internal representation. Several methods for detecting inconsistent internal states were proposed and demonstrated in Meliora, and related work by Chapman and Kaelbling [Chapman and Kaelbling, 1991] and Tan [Tan, 1991a] was described.

Next, the effect of unbiased search on the learning rate was addressed. It was shown that when an agent has little or no prior knowledge of a task and when reward is sparse or delayed, lack of initial guidance combined with lack of feedback can lead to unstructured searches and excessive learning times. Two cooperative mechanisms, Learning-with-an-External-Critic (LEC) and Learning-By-Watching (LBW), that reduce search were described and analyzed. Formal analysis showed that for a restricted class of decision problems, these algorithms have expected learning times that are dependent on the length of the optimal solution path and independent of the state space size. Experimental results on the GB-task confirm these results and showed LEC and LBW to improve significantly the performance of Meliora. Part of the performance improvement was due to a reduction in the latency between overt action and feedback; however, the additional feedback available through these mechanisms was also used to detect inconsistent internal states more quickly, facilitating identification.

The work described in this dissertation only partially achieves our objective to develop algorithms for adaptive perception and action. The decision to split perceptual and overt control into separate stages precludes the application of the CR-method to tasks that require overt actions for identification. Moreover, even when the CR-method is applicable, there exist tasks that it cannot solve optimally. Nevertheless, the CR-method represents initial progress towards a theory of adaptive control that incorporates active perception. Furthermore, it appears that the notion of consistency can be used to extend the CR-method and derive alternative methods that address some of the current limitations. The Q-CUP algorithm and the schema-based approach to task decomposition are examples of two such directions for future research.

Bibliography

[Agre, 1985] Philip E. Agre, "Routines," Technical Report 828, MIT AI Memo, 1985.

[Agre, 1988] Philip E. Agre, *The Dynamic Structure of Everyday Life*, PhD thesis, MIT Artificial Intelligence Lab., 1988, (Tech Report No. 1085).

[Agre, ming] Philip E. Agre, *The Dynamic Structure of Everyday Life*, Cambridge University Press, Cambridge, forthcoming.

[Agre and Chapman, 1987] Philip E. Agre and David Chapman, "Pengi: an implementation of a theory of activity," In *AAAI*, pages 268-272, 1987.

[Aloimonos *et al.*, 1987] J. Aloimonos, A. Bandopadhay, and I. Weiss, "Active vision," In *Proceedings of the First IEEE International Conference on Computer Vision*, pages 35-54, June 1987, (Also in International Journal of Computer Vision, 1, 4, pages 333-356, 1988.).

[Anderson, 1986] Charles W. Anderson, *Learning and Problem Solving with Multilayer Connectionist Systems*, PhD thesis, University of Massachusetts, Amherst, 1986.

[Bajcsy and Allen, 1984] R. Bajcsy and P. Allen, "Sensing strategies," In *U.S.-France Robotics Workshop*, Univ. of Pennsylvania, Philadelphia, PA, November 1984.

[Ballard, 1989a] D. H. Ballard, "Behavioral constraints on computer vision," *Image and Vision Computing*, 7(1), February 1989.

[Ballard, 1991] D. H. Ballard, "Animate vision," *Artificial Intelligence*, 48:57-86, 1991.

[Ballard and Ozcandarli, 1988] D. H. Ballard and A. Ozcandarli, "Eye fixation and early vision: kinetic depth," In *Proceedings of the Second IEEE International Conference on Computer Vision*, December 1988.

[Ballard, 1989b] Dana H. Ballard, "Reference frames for animate vision," In *Proceedings of the Eleventh IJCAI*, pages 1635–1641, Detroit, MI, 1989.

[Barto *et al.*, 1991] A.B. Barto, S.J. Bradtke, and S.P. Singh, "Real-Time learning and control using asynchronous dynamic programming," Technical Report 91-57, University of Massachusetts, Amherst, MA, 1991.

[Barto *et al.*, 1990] Andrew B. Barto, Richard S. Sutton, and Chris Watkins, "Sequential decision problems and neural networks," In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*. Morgan Kaufmann, San Mateo, CA, 1990.

[Barto *et al.*, 1983] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson, "Neuron-like elements that can solve difficult learning control problems," *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983.

[Bellman, 1957] R. E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[Bertsekas, 1987] D. P. Bertsekas, *Dynamic Programming: Deterministic and Stochastic Models*, Prentice-Hall, 1987.

[Booker, 1988] Lashon B. Booker, "Classifier systems that learn internal world models," *Machine Learning*, 3:161–192, 1988.

[Bradley, 1968] J. V. Bradley, *Distribution Free Statistical Tests*, Prentice Hall Inc., Englewood Cliffs, New Jersey, 1968.

[Bradt *et al.*, 1956] R. N. Bradt, S. M. Johnson, and S. Karlin, "On sequential designs for maximizing the sum of n observations," *Annals of Math. Statist.*, 27:1060–1074, 1956.

[Bradt and Karlin, 1956] R. N. Bradt and S. Karlin, "On the design and comparison of certain dichotomous experiments," *Annals of Math. Statist.*, 27:390–309, 1956.

[Brooks, 1986] Rodney A. Brooks, "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, pages 14–22, April 1986.

[Chapman, 1989] David Chapman, "Penguins can make cake," *AI Magazine*, 10(4):45–50, 1989.

[Chapman, 1990a] David Chapman, "Intermediate vision: architecture, implementation, and use," Technical Report 90-06, Teleos Research, Palo Alto, CA, 1990.

[Chapman, 1990b] David Chapman, *Vision, Instruction, and Action*, PhD thesis, MIT Artificial Intelligence Laboratory, 1990, (Technical Report 1204).

[Chapman and Kaelbling, 1991] David Chapman and Leslie Pack Kaelbling, "Learning from delayed reinforcement in a complex domain," In *Proceedings of IJCAI*, 1991, (Also Teleos Technical Report TR-90-11, 1990).

[Chrisman and Simmons, 1991] L. Chrisman and R. Simmons, "Sensible planning: focusing perceptual attention," In *Proceedings of the Tenth National Conference on Artificial Intelligence*, July 1991.

[Dean and Wellman, 1991] Thomas Dean and Michael Wellman, *Planning and Control*, Morgan Kaufmann, San Mateo, CA, 1991.

[Dickmanns, 1989] E. D. Dickmanns, "Real-time machine vision exploiting integral spatio-temporal world models," In *11th International Joint Conference on Artificial Intelligence*, Detroit, MI, August 1989.

[Dubins and Savage, 1965] L. Dubins and L. Savage, *How to Gamble If You Must: Inequalities for Stochastic Processes*, McGraw-Hill, New York, 1965.

[Epstein, 1967] Richard Epstein, *The Theory of Gambling and Statistical Logic*, Academic Press, New York, 1967.

[Feldaman, 1962] D. Feldaman, "Contributions to the two-armed bandit problem," *Annals of Math. Statist.*, 33(2):847–856, 1962.

[Fikes *et al.*, 1972] Richard E. Fikes, Paul E. Hart, and Nils J. Nilsson, "Learning and executing generalized robot plans," *Artifical Intelligence*, 3(4):251–288, 1972, (Also in Readings in Artificial Intelligence, 1980).

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson, "STRIPS: a new approach to the application of theorem proving to problem solving," *Artificial Intelligence*, 2:189–208, 1971.

[Franklin, 1988] Judy A. Franklin, "Refinement of robot motor skills through reinforcement learning," In *Proceedings of the 27th IEEE Conference on Decision and Control*, Austin, TX, December 1988.

[Garvey, 1976] T. D. Garvey, "Perceptual strategies for purposive vision," Technical Report 117, SRI International, September 1976.

[Gibson, 1979] J. J. Gibson, *The Ecological Approach to Visual Perception*, Houghton Mifflin, Boston, 1979.

[Ginsberg, 1989] Matthew L. Ginsberg, "Universal planning: an (almost) universally bad idea," *AI Magazine*, 10(4):41–44, 1989.

[Grefenstette, 1988] John J. Grefenstette, "Credit assignment in rule discovery systems based on genetic algorithms," *Machine Learning*, 3:225–245, 1988.

[Hartman, 1990] Leo Hartman, *Decision Theory and The Cost of Planning*, PhD thesis, Dept. of Computer Science, University of Rochester, 1990.

[Hayes, 1973] Patrick Hayes, "The frame problem and related problems in artificial intelligence," In A. Elithorn and D. Jones, editors, *Artificial and Human Thinking*, pages 45–49. San Francisco: Jossey-Bass, 1973, (Also in Readings In Artificial Intelligence, 1980).

[Hirai and Sato, 1989] S. Hirai and T. Sato, "Motion understanding for world model manipulation of telerobot," In *Proceedings of the 5th International Symposium on Robotics Research*, pages 124–131, 1989.

[Holland and Reitman, 1978] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," In D. A. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems*. Academic Press, New York, 1978.

[Holland, 1973] John H. Holland, "Genetic algorithms and the optimal allocation of trials," *SIAM Journal of Computing*, 2(2):88–105, 1973.

[Holland, 1975] John H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[Holland *et al.*, 1986] John H. Holland, Keith F. Holyoak, Richard E. Nisbett, and Paul R. Thagard, *Induction: Processes of Inference, Learning, and Discovery*, MIT Press, 1986.

[Jordan and Rumelhart, 1990] Michael I. Jordan and David E. Rumelhart, "Supervised learning with a distal teacher," Technical report, MIT, 1990.

[Julesz, 1981] B. Julesz, "Textons, the elements of texture perception and their interactions," *Nature*, 290:91–97, 1981.

[Kaelbling, 1987] L. P. Kaelbling, "An architecture for intelligent reactive systems," In *Reasoning About Actions and Plans: Proceedings of the 1986 Workshop*. Morgan Kaufmann, 1987.

[Kaelbling, 1989] Leslie P. Kaelbling, "A formal framework for learning in embedded systems," In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 350–353, 1989.

[Kaelbling, 1990] Leslie P. Kaelbling, *Learning in Embedded Systems*, PhD thesis, Stanford University, 1990.

[Kautz, 1987] Henry A. Kautz, *A Formal Theory of Plan Recognition*, PhD thesis, Department of Computer Science, University of Rochester, 1987.

[Kuniyoshi *et al.*, 1990] Yasuo Kuniyoshi, Hirochika Inoue, and Masayuki Inaba, "Design and implementation of a system that generates assembly programs from visual recognition of human action sequences," In *Proceedings of the IEEE International Workshop on Intelligent Robots and Systems*, pages 567–574, 1990.

[Kyburg, 1991] Henry Kyburg, "Evidential probability," In *Proceedings of the Twelveth International Conference on Artificial Intelligence*, 1991.

[Laird *et al.*, 1986] John E. Laird, Paul S. Rosenbloom, and Alan Newell, "Chunking in SOAR: the anatomy of a general learning mechanism," *Machine Learning*, 1(1):11–46, 1986.

[Lehmann, 1959] E. L. Lehmann, *Testing Statistical Hypothesis*, John Wiley and Sons, London, 1959.

[Lin, 1990] Long-Ji Lin, "Self-improving reactive agents: case studies of reinforcement learning frameworks," In *Proceedings of the First International Conference on the Simulation of Adaptive Behavior*, September 1990.

[Lin, 1991] Long Ji Lin, "Self-improvment based on reinforcement learning, planning, and teaching," In *Proceedings of the Eighth International Workshop on Machine Learning*, 1991.

[Mahadevan and Connell, 1991] Sridhar Mahadevan and Jonathan Connell, "Scaling reinforcement learning to robotics by exploiting the subsumption architecture," In *Proceedings of the Eighth International Workshop on Machine Learning*, 1991.

[Marr, 1976] David Marr, "Early processing of visual information," *Phil. Trans. Roy. Soc. Lond. B*, 275:483–524, 1976.

[Marr and Nishihara, 1978] David Marr and Keith Nishihara, "Representation and recognition of the spatial organization of three dimensional shapes," *Proc. Roy. Soc. Lond. B*, 200:269–291, 1978.

[McCallum, 1991] Andrew McCallum, personal communication, 1991.

[McCarthy, 1977] John McCarthy, "Epistemological problems of artificial intelligence," In *IJCAI-5*, pages 1038–1044, 1977, (Also in Readings in Artificial Intelligence).

[McCarthy and Hayes, 1969] John McCarthy and Patrick Hayes, "Some philosophical problems from the standpoint of artifical intelligence," In *Machine Intelligence 4*. Edinburgh: Edinburgh University Press, 1969, Also in Readings in Artificial Intelligence.

[Michie and Chambers, 1968] D. Michie and R. Chambers, "BOXES: An experiment in adaptive control.," In E. Dale and D. Michie, editors, *Machine Intelligence 2*, pages 137–152. Oliver and Boyd, Edinburgh, 1968.

[Minsky, 1954] Marvin L. Minsky, *Theory of Neural-Analog Reinforcement Systems and Its Application to The Brain-Model Problem*, PhD thesis, Princeton University, 1954.

[Mozer and Bachrach, 1989] M. Mozer and J. Bachrach, "Discovering the structure of a reactive environment by exploration," Technical Report 451-89, Dept. of Computer Science, Univ. of Colorado, 1989.

[Newtson *et al.*, 1977] D. Newtson, G. Engquist, and J. Bois, "The objective basis of behavior units," *Journal of Personality and Social Psychology*, 35(12):847–862, 1977.

[Nguyen and Widrow, 1989] D. Nguyen and B. Widrow, "The truck backer-upper: An example of self-learning in neural networks," In *Proceedings of the First International Joint Conference on Neural Networks*, 1989.

[Nilsson, 1980] Nils Nilsson. *Principles of Artificial Intelligence*, Tioga, Palo Alto, CA, 1980.

[Noton, 1970] D. Noton, "A theory of visual pattern perception," *IEEE Transactions on Systems, Science & Cybernetics*, 6:349–357, October 1970.

[Noton and Stark, 1971a] D. Noton and L. Stark, "Eye movements and visual perception," *Scientific American*, 224(6):33–43, 1971.

[Noton and Stark, 1971b] D. Noton and L. Stark, "Scanpaths in saccadic eye movements while viewing and recognizing patterns," *Vision Research*, 11(929), 1971.

[O'Regan and Levy-Schoen, 1983] J. K. O'Regan and A. Levy-Schoen, "Integrating visual information from successive fixations: Does trans-saccadic fusion exist?," *Vision Research*, 28(8):765–768, 1983.

[Pearl, 1988] Judea Pearl, *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, San Mateo, CA, 1988.

[Pineda, 1987] F. J. Pineda, "Generalization of backpropagation to recurrent and higher order networks," In *Proceedings of the IEEE Conference on Neural Information Processing Systems*, 1987.

[Randles and Wolfe, 1979] R. H. Randles and D. A. Wolfe, *Introduction to the Theory of Nonparametric Statistics*, John Wiley and Sons, New York, 1979.

[Rimey and Brown, 1990] R. Rimey and C. Brown, "Selective attention as sequential behavior: Modelling eye movements with an augmented hidden markov model," Technical Report 327, Dept. of Computer Science, University of Rochester, Rochester, NY, 1990.

[Riolo, 1988] Rick L. Riolo, *Empirical Studies of Default Heirarchies and Sequences of Rules in Learning Classifier Systems*, PhD thesis, Dept. of Computer Science and Engineering, University of Michigan, 1988.

[Rivest and Schapire, 1987] R. Rivest and R. Schapire, "A new approach to unsupervised learning in deterministic environments," In P. Langley, editor, *Proceedings of the fourth international workshop on machine learning*, pages 364–375, Irvine, California, 1987.

[Robbins, 1952] H. Robbins, "Some aspects of the sequential design of experiments," *Bull. Amer. Math. Soc.*, 58:527–535, 1952.

[Roberts, 1984] Fred S. Roberts, *Applied Combinatorics*, Prentice Hall, Englewood Cliffs, NJ, 1984.

[Romanycia, 1987] Marc Romanycia, "The design and control of visual routines for the computation of simple geometric properties and relations," Technical Report 87-34, Dept. of Computer Science, University of British Columbia, 1987.

[Romanycia, 1988] Marc Romanycia, "Visual Routines: ingredients, design, and control," Unpublished paper, Dept. of Computer Science, University of British Columbia, 1988.

[Ross, 1983] S. Ross, *Introduction to Stochastic Dynamic Programming*, Academic Press, New York, NY, 1983.

[Sacerdoti, 1977] E. D. Sacerdoti, *A Structure for Plans and Behavior*, New York: Elsevier, 1977.

[Samuel, 1963] A. L. Samuel, "Some studies in machine learning using the game of checkers," In E. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 71–105. Krieger, Malabar, FL, 1963.

[Schmidhuber, 1990a] Jurgen Schmidhuber, "Learning to generate focus trajectories for attentive vision," Technical Report Report FKI-128-90, Technische Universitat Munchen, 1990.

[Schmidhuber, 1990b] Jurgen Schmidhuber, "Making the world differentiable: on using self-supervised fully recurrent neural networks for dynamic reinforcement learning and planning in non-stationary environments," Technical Report Report FKI-126-90 (revised), Technische Universitat Munchen, 1990.

[Schmidhuber, 1990c] Jurgen Schmidhuber, "Networks adjusting networks," Technical Report FKI-125-90, Technische Universitat Munchen, 1990.

[Schoppers, 1989a] Marcel J. Schoppers, "In defense of reaction plans as caches," *AI Magazine*, 10(4):51–60, 1989.

[Schoppers, 1989b] Marcel J. Schoppers, *Representation and Automatic Synthesis of Reaction Plans*, PhD thesis, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1989.

[Shafer, 1990] Steve Shafer, "Why we can't model the physical world," (for the 25th anniversary of the CMU CS Dept.), September 1990.

[Simard, 1991] Patrice Simard, *Learning State Space Dynamics in Recurrent Networks*, PhD thesis, Dept. of Computer Science, University of Rochester, 1991.

[Simmons, 1990] R. Simmons, "Robust behavior with limited resources," In *AAAI Stanford Spring Symposium*, 1990.

[Singh, 1991] Satinder Singh, "Transfer of learning across compositions of sequential tasks," In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 348–352. Morgan Kaufmann, 1991.

[Sussman, 1975] Gerald Sussman, *A Computer Model of Skill Acquistion*, American Elsevier, New York, 1975.

[Sutton, 1984] Richard S. Sutton, *Temporal Credit Assignment In Reinforcement Learning*, PhD thesis, University of Massachusetts at Amherst, 1984, (Also COINS Tech Report 84-02).

[Sutton, 1988] Richard S. Sutton, "Learning to predict by the method of temporal differences," *Machine Learning*, 3(1):9–44, 1988.

[Sutton, 1990a] Richard S. Sutton, "First results with DYNA, an integrated architecture for learning, planning, and reacting," In *Proceedings of the AAAI Spring Symposium on Planning in Uncertain, Unpredictable, or Changing Environments*, 1990.

[Sutton, 1990b] Richard S. Sutton, "Integrating architectures for learning, planning, and reacting based on approximating dynamic programming," In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, TX, 1990. Morgan Kaufmann.

[Sutton, 1991] Richard S. Sutton, "Planning by incremental dynamic programming," In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 353-357. Morgan Kaufmann, 1991.

[Sutton and Pinette, 1985] Richard S. Sutton and Brian Pinette, "The learning of world models by connectionist networks," In *Proceedings of the Seventh Annual Conf. of the Cognitive Science Society*, pages 54-64, 1985.

[Swain, 1990] M. Swain, *Color Indexing*, PhD thesis, Dept. of Computer Science, University of Rochester, 1990, (Also Technical Report # 360).

[Tan, 1991a] Ming Tan, "Cost sensitive reinforcement learning for adaptive classification and control," In *Proceedings of the Ninth International Conference on Artificial Intelligence*, 1991.

[Tan, 1991b] Ming Tan, *Cost Sensitive Robot Learning*, PhD thesis, Carnegie Mellon University, 1991.

[Tan and Schlimmer, 1990] Ming Tan and Jeffery C. Schlimmer, "Two case studies in cost-sensitive concept acquisition," In *Proceedings of AAAI-90*, 1990.

[Thibadeau, 1986] R. Thibadeau, "Artificial perception of actions," *Cognitive Science*, 10(2):117-149, 1986.

[Thrun and Moller, 1991] S. Thrun and K. Moller, "Planning with an adaptive world model," In D. S. Tourtezky and R Lippmann, editors, *Advances in Neural Information Processing Systems 3*. Morgan Kaufmann, 1991.

[Treismann and Gelade, 1980] Anne Treismann and Garry Gelade, "A feature-integration theory of attention," *Cognitive Psychology*, 12:97-136, 1980.

[Tsotsos, 1987] J. Tsotsos, "A complexity level analysis of vision," In *Proceedings of IJCCV*, June 1987.

[Tsuji *et al.*, 1977] S. Tsuji, A. Morizono, and S. Kuroda, "Understanding a simple cartoon film by a computer vision system," In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, pages 609-610, 1977.

[Ullman, 1984] Shimon Ullman, "Visual Routines," *Cognition*, 18:97-159, 1984, (Also in: Visual Cognition, S. Pinker ed., 1985).

[Watkins, 1989] Chris Watkins, *Learning from delayed rewards*, PhD thesis, Cambridge University, 1989.

[Watkins and Dayan, 1992] Christopher Watkins and Peter Dayan, "Q-Learning," *Machine Learning*, 1992, (to appear).

[Werbos, 1987] Paul J. Werbos, "Building and understanding adaptive systems: A statistical/numerical approach to factory automation and brain research," *IEEE Transations on Systems, Man, and Cybernetics*, 1987.

[Whitehead, 1989] Steven D. Whitehead, "Scaling in Reinforcement Learning," Technical Report TR 304, Computer Science Dept., University of Rochester, 1989.

[Whitehead, 1991] Steven D. Whitehead, "Complexity and cooperation in reinforcement learning," In *Proceedings of the Tenth National Conference on Artificial Intelligence*, 1991, (A similar version also appears in the Proceedings of the Eighth International Workshop on Machine Learning, Evanston, IL, June 1991.).

[Whitehead and Ballard, 1989a] Steven D. Whitehead and Dana H. Ballard, "Reactive behavior, learning, and anticipation," In *Proceedings of the NASA Conference on Space Telerobotics*, Pasadena, CA, 1989.

[Whitehead and Ballard, 1989b] Steven D. Whitehead and Dana H. Ballard, "A role for anticipation in reactive systems that learn," In *Proceedings of the Sixth International Workshop on Machine Learning*, Ithaca, NY, 1989. Morgan Kaufmann.

[Whitehead and Ballard, 1990] Steven D. Whitehead and Dana H. Ballard, "Active perception and reinforcement learning," *Neural Computation*, 2(4), 1990, (Also In the Proceedings of the Seventh International Conference on Machine Learning, Morgan Kaufmann, June 1990).

[Whitehead and Ballard, 1991a] Steven D. Whitehead and Dana H. Ballard, "Learning to Perceive and Act by Trial and Error," *Machine Learning*, 7(1), 1991, (Also Tech. Report # 331, Department of Computer Science, University of Rochester, 1990.).

[Whitehead and Ballard, 1991b] Steven D. Whitehead and Dana H. Ballard, "A study of cooperative mechanisms for faster reinforcement learning," TR 365, Computer Science Dept., University of Rochester, Feburary 1991.

[Williams and Zipser, 1988] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent networks," Technical Report 8805, Univ. of California, San Diego, 1988.

[Williams, 1987] Ronald J. Williams, "Reinforcement-learning connectionist systems," Technical Report NU-CCS-87-3, College of Computer Science, Northeastern University, Boston, MA, 1987.

[Wixson, 1991] Lambert Wixson, "Scaling reinforcement learning techniques via modularity," In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 368–372. Morgan Kaufmann, 1991.

[Wixson, 1990] Lambert E. Wixson, "Real-time qualitative detection of multicolored objects for object search," In *DARPA Image Understanding Workshop*, September 1990.

[Wixson and Ballard, 1991] Lambert E. Wixson and Dana H. Ballard, "Learning Efficient Sensing Sequences for Object Search," In *AAAI Fall Symposium*, November 1991.

[Yarbus, 1967] A.L. Yarbus, *Eye Movements and Vision*, Plenum Press, 1967.

[Yee *et al.*, 1990] Richard C. Yee, Sharad Saxena, Paul E. Utgoff, and Andrew G. Barto, "Explaining temporal-differences to create useful concepts for evaluating states," In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 1990.

A Proofs for Search Analysis Theorems

Following are the proofs for the theorems given in Chapter 7.

Theorem 3: *In a homogeneous state space, the expected time needed by a* zero-initialized Q-learning *system to learn the actions along an optimal solution path for a problem solving task is bounded below by the expression*

$$c_1 * \left\{ c_2 \left[\frac{P_+}{P_-}\right]^{k-i} \left[\left(\frac{P_+}{P_-}\right)^i - 1\right] + \frac{i}{1-2P_+} \right\} \tag{A.1}$$

where

$$c_1 = \left(\frac{1}{1-P_=}\right), \tag{A.2}$$

$$c_2 = \frac{P_+}{(1-2P_+)^2}, \tag{A.3}$$

$$P_= = \frac{b_=}{b_= + b_+ + b_-}, \tag{A.4}$$

$$P_+ = \frac{b_+}{b_+ + b_-}, \tag{A.5}$$

and

$$P_- = 1 - P_+, \tag{A.6}$$

and where i is the length of the optimal solution, and k is the depth bound on the state space (with respect to the goal).

Proof:

There are two keys to this proof. The first is to recognize that on its initial trial a zero-initialized Q-learning system performs a pure random walk that begins in the start state and ends in the goal state. The second is to recognize that the expected length of a random walk on a k-bounded homogeneous state space is the same as the expected length of the random walk on an equivalent 1-dimensional state space. That is, because all states (except the boundaries) have the same connectivity pattern, states whose distance to the goal are the same can be collapsed into a single node. Similarly, all boundary states can be collapsed into the same node. Thus, results for the expected search time required by a zero-initialized Q-learning system can be obtained by analyzing random walks on a 1-dimensional state space.

Let's begin with some notation. Let b_-, b_+, and $b_=$ be the number of actions that, in any given state, cause the system to decrease, increase, and leave the distance to the goal unchanged, respectively. Define P_+ and P_- as the conditional probability that the system, when choosing randomly, takes an action that increases or decreases the distance to the goal, respectively, given that a distance changing action is chosen. Also, define $P_=$ as the unconditional probability that the system chooses an action that leaves the distance unchanged. Then,

$$P_+ = \frac{b_+}{b_+ + b_-}, \tag{A.7}$$

$$P_- = 1 - P_+, \tag{A.8}$$

and

$$P_= = \frac{b_=}{b_= + b_+ + b_-}. \tag{A.9}$$

Finally, let $E^k_{i,j}$ denote the expected length of a random walk on a bounded $(k+1)$ length 1-dimensional state space that begins in state i and ends when the system first encounters state j. Also, assume the states in the state space are numbered consecutively, 0, 1, 2, ..., $k-1$, k.

A closed form expression for $E^k_{i,0}$ can be obtained by expressing $E^k_{k,0}$ recursively and solving the recurrence. To do so, we momentarily assume $P_= = 0$ and make the following observations. First, notice that for $i, j > 0$, $0 \leq k, k-i \leq n$ and $0 \leq k-j, k-i-j \leq n-j$

$$E^n_{k,k-i} = E^{n-j}_{k-j,k-i-j}. \tag{A.10}$$

This follows since the regions of the state spaces involved in these two random walks are homomorphic. Also notice that in general for $i \leq k \leq j$

$$E^n_{i,j} = E^n_{i,k} + E^n_{k,j}. \tag{A.11}$$

Finally, note that in general $E^n_{i,i} = 0$.

Using Equation A.10 and Equation A.11 we have, for $k \geq 1$

$$\begin{aligned} E^k_{k,0} &= E^k_{k,1} + E^k_{1,0} & \text{(A.12)} \\ &= E^{k-1}_{k-1,0} + E^k_{1,0}. & \text{(A.13)} \end{aligned}$$

Next, an expression for $E^k_{1,0}$, in terms of expectations for a $k-1$ state space can be obtained by expanding the expectation. For $k > 1$,

$$E^k_{1,0} = 1 * P_- + [1 + E^k_{2,0}] * P_+. \tag{A.14}$$

But by Equation A.10 & A.11,

$$E^k_{2,0} = E^k_{2,1} + E^k_{1,0} \tag{A.15}$$
$$= E^{k-1}_{1,0} + E^k_{1,0}. \tag{A.16}$$

Thus, Equation A.14 can be rewritten as

$$E^k_{1,0} = P_- + P_+[1 + E^{k-1}_{1,0} + E^k_{1,0}], \tag{A.17}$$

or after separating terms,

$$E^k_{1,0} = \frac{1 + P_+ E^{k-1}_{1,0}}{1 - P_+}. \tag{A.18}$$

Returning to Equation A.13, we have for $k > 1$

$$E^k_{k,0} = E^{k-1}_{k-1,0} + \frac{1 + P_+ E^{k-1}_{1,0}}{1 - P_+}. \tag{A.19}$$

Next, notice that by Equation A.11, for $k > 1$

$$E^{k-1}_{k-1,0} = E^{k-1}_{k-1,1} + E^{k-1}_{1,0}. \tag{A.20}$$

Which by Equation A.10 leads to

$$E^{k-1}_{k-1,0} = E^{k-2}_{k-2,0} + E^{k-1}_{1,0} \tag{A.21}$$

or solving for $E^{k-1}_{1,0}$,

$$E^{k-1}_{1,0} = E^{k-1}_{k-1,0} - E^{k-2}_{k-2,0}. \tag{A.22}$$

Substituting Equation A.22 into Equation A.19 yields a recursive definition for $E^k_{k,0}$,

$$E^k_{k,0} = E^{k-1}_{k-1,0} + \frac{1 + P_+[E^{k-1}_{k-1,0} - E^{k-2}_{k-2,0}]}{1 - P_+}. \tag{A.23}$$

Factoring terms yields

$$E^k_{k,0} = \left(\frac{1}{1 - P_+}\right) E^{k-1}_{k-1,0} - \left(\frac{P_+}{1 - P_+}\right) E^{k-2}_{k-2,0} + \frac{1}{1 - P_+}. \tag{A.24}$$

Initial conditions for the recurrence can be obtained by noticing that

$$E^0_{0,0} = 0 \tag{A.25}$$

and by expanding the expectation

$$E^1_{1,0} = (1 + E^1_{1,0})P_+ + (1)P_-, \tag{A.26}$$

which when solving for $E^1_{1,0}$ yields

$$E^1_{1,0} = \frac{1}{1-P_+}. \tag{A.27}$$

To simplify notation, let $a_k = E^k_{k,0}$, $c_1 = \frac{1}{1-P_+}$, and $c_0 = \frac{-P_+}{1-P_+}$. Then for $k > 1$, Equation A.24 can be rewritten as

$$a_{k+2} = c_1 a_{k+2} + c_0 a_k + c_1, \tag{A.28}$$

where $a_0 = 0$ and $a_1 = \frac{1}{1-P_+}$. Equation A.28 is a non-homogeneous linear recurrence equation that can be solved using generating functions [Roberts, 1984]. In particular, solving the recurrence [Whitehead, 1991] yields

$$E^k_{k,0} = \frac{P_+}{(1-2P_+)^2}\left[\frac{P_+}{P_-}\right]^k + \frac{(k+1)}{(1-2P_+)} - \frac{(1-P_+)}{(1-2P_+)^2}. \tag{A.29}$$

A general expression for $E^k_{i,0}$ can be obtained from Equation A.29 by noting that

$$E^k_{i,0} = E^k_{k,0} - E^k_{k,i} = E^k_{k,0} - E^{k-i}_{k-i,0}, \tag{A.30}$$

which when combined with Equation A.29 yields

$$E^k_{i,0} = \frac{P_+}{(1-2P_+)^2}\left[\frac{P_+}{1-P_+}\right]^{k-i}\left[\left(\frac{P_+}{P_-}\right)^i - 1\right] + \frac{i}{1-2P_+}. \tag{A.31}$$

Recall that the derivation of Equation A.31 is based on the assumption that $P_= = 0$. An expression for $E^k_{i,0}$ for the general case where $P_= \neq 0$ can be obtained by noticing that $E^k_{i,0}$ in the general case is just Equation A.31 multiplied by the expected time needed to take an action that changes the distance to the goal. Because action selections are independent, the time needed to choose a "distance changing action" is a geometric random variable with mean $1/(1-P_=)$. Thus,

$$E^k_{i,0} = \frac{1}{1-P_=} * \left\{\frac{P_+}{(1-2P_+)^2}\left[\frac{P_+}{1-P_+}\right]^{k-i}\left[\left(\frac{P_+}{P_-}\right)^i - 1\right] + \frac{i}{1-2P_+}\right\}. \tag{A.32}$$

Equation A.32 is the expected search time needed by a zero-initialized Q-learning system to first solve a task on a homogeneous k-bounded state space that begins i steps from the goal state. For 1-step Q-learning, the system will have learned at most the last step along the optimal solution. Q-learning algorithms that use multi-step estimators may learn more steps along the optimal path. In fact, it is possible (albeit exceedingly unlikely) that they will correctly learn all the steps along an optimal solution path in one trial. In any case, a zero-initialized Q-learner must solve at least the first trial by random search. Thus,

Equation A.32 provides a lower bound on the expected learning time.
□

Theorem 6: *The expected time needed by a zero-initialized BB-LEC system to learn the actions along an optimal path for a homogeneous state space of size k is bounded above by*

$$\frac{k}{P_{\text{critic}}} * |S| * b \tag{A.33}$$

where P_{critic} is the probability that on a given step the external critic provides feedback, $|S|$ is the total number of states in the state space, b is the branching factor (or total number of possible actions per state) and k is the depth of the state space.

Proof:

The proof is based on the idea that, because the state space is recurrent, the system will eventually either solve the task or receive feedback from the critic for every possible state-action pair in the state space. By subsumption, the system will have learned (i.e., be appropriately biased with respect to) the actions along an optimal solution path once it has received feedback for all possible state-action pairs. Thus, a weak upper bound on the expected time needed to learn an optimal policy for states along an optimal path can be obtained by determining the expected time needed to receive feedback for every state-action pair, given that the agent has not learned the optimal actions along an optimal trajectory first.

Let $n_{knw}(t)$ denote the number of state-action pairs for which the system has received feedback from the critic by time t. We will say that a state-action pair is *known* if when the system tried the pair it received feedback from the critic. Notice that $n_{knw}(0) = 0$, and n_{knw} monotonically increases in time, as the system receives feedback for new state-action pairs. Since there are at most $|S| * b$ state-action pairs, we would like to know the expected time required for $n_{knw}(t) = |S| * b$.

This can be obtained by determining the rate at which n_{knw} is expected to increase. n_{knw} increases at the expected rate of at least P_{critic}/k. That is, on average, the system can expect to receive feedback for an *unknown* state-action pair at least every k/P_{critic} steps. This follows since in any sequence of k consecutive steps the system must either solve the task (i.e., take k correct actions) or take an action that is non-optimal. But the system will take a non-optimal step only if the corresponding state-action pair is unknown. Thus, every k steps the system either solves the task or tries an unknown state-action pair. Finally, since the critic provides feedback with probability P_{critic}, it takes on average at most $k * 1/P_{critic}$ steps before the system tries an unknown state-action pair and receives feedback from the critic.

Thus, the expected time for $n_{knw}(t) = |S| * b$, is at most

$$\frac{|S| * b}{\frac{P_{critic}}{k}} \tag{A.34}$$

or

$$\frac{k}{P_{critic}} * |S| * b. \tag{A.35}$$

□

Theorem 7: ***If a zero-initialized BB-LEC system has access to an inverse model, then the expected time needed to learn the actions along an optimal path for a homogeneous state space is linear in the solution length i, independent of state space size, and bounded above by the expression***

$$\left[\frac{2}{P_{-}(1 - P_{=})} - 1\right] * i. \tag{A.36}$$

Proof:
This theorem is based on the idea that with an inverse model and an external critic, the agent can systematically search for the optimal action in every state. That is, it can try an action, determine from the critic if it was incorrect, and if so take the inverse and try again or else proceed to determine the next optimal action.

The expression in Equation A.36 is just i, the length of the optimal solution, multiplied by the expected number of steps required to determine the optimal action in a given state. Since for zero-initialized Q-learning, we assume ties are broken by selecting randomly, the number of action-inverse cycles that must be tried before the optimal action is performed is at most the expected value of a geometric random variable with parameter $P_{-}(1 - P_{=})$. Since all but the last cycle require 2 steps, the expected number of steps required to take one optimal action is less than $\frac{2}{P_{-}(1-P_{=})} - 1$, and the total number of steps required is on average less than

$$\left[\frac{2}{P_{-}(1 - P_{=})} - 1\right] * i. \tag{A.37}$$

□

The above proof is for $p_{critic} = 1.0$. For $p_{critic} < 1.0$, Equation A.36 must be multiplied by a factor of $1/p_{critic}$.

Theorem 8: *A zero-initialized BB-LEC system that aborts a trial and starts anew if it fails to solve the task after n_q ($n_q \geq i$) steps has, for a homogeneous* *problem solving task, an expected initial solution time that is linear in i, independent of state space size, and bounded from above by the expression*

$$\left(\frac{1}{P_{-}(1-P_{=})}\right) * n_q i. \qquad (A.38)$$

Proof:
This proof is similar to the last and is based on the idea that by aborting a trial after n_q steps the system can return to the site of previous decisions and systematically discover the optimal action. While in Theorem 7 the system used an explicit inverse model, in this theorem the inverse is performed implicitly.

As before, the system must learn the optimal action for the i steps along the optimal solution path. The expected time needed to learn each optimal step is just the expected number of times the system must cycle through this inversion loop multiplied by the expected length of the cycle. In the worst case, the expected length of the inversion cycle is n_q steps. The expected number of cycles required for each state along an optimal trajectory is less than the expected value of a geometric random variable with parameter $P_{-}(1-P_{=})$. Thus, the expected number of steps required to learn the steps along an optimal path is less than

$$\left(\frac{1}{P_{-}(1-P_{=})}\right) * n_q i. \qquad (A.39)$$

□

Again, for $p_{critic} < 1.0$, Equation A.38 must be multiplied by a factor of $1/p_{critic}$.

Theorem 10: *The expected time required for a population of naive (zero-initialized) Q-learning agents using LBW to learn the actions along an optimal path decreases to the minimum required learning time at a rate that is $\Omega(1/n)$, where n is the size of the population.*

Proof:
Let E_n denote the expected time required for one of the n zero-initialized Q-learning agents to solve the task for the first time. Let $P_{n,k}$ be the probability that one of the n agents first solves the task in the kth round. Recall, we assume agents operate in parallel. After the kth round each agent has taken k steps. Thus,

$$E_n = \sum_{k=o}^{\infty} k \; P_{n,k} \qquad (A.40)$$

where o is the length of the optimal solution path.

In general, each agent behaves independently (because each is zero-initialized and each performs a random walk), so

$$P_{n,k} = \sum_{i=1}^{n}(1 - P_{1,k})^{i-1}P_{1,k} \tag{A.41}$$

$$= 1 - (1 - P_{1,k})^n. \tag{A.42}$$

Returning to Equation A.40 we have,

$$E_n = o * P_{n,o} + (1 - P_{n,o}) * \sum_{k=o+1}^{\infty} kP_{n,k|\neg o} \tag{A.43}$$

where $P_{n,k|\neg o}$ is the probability that one of the n agents first solves the task in the kth round, given that they have all failed to do so in the oth round. Now, for all $n \geq 1$

$$\sum_{k=o+1}^{\infty} kP_{n,k|\neg o} \leq \sum_{k=o+1}^{\infty} kP_{1,k|\neg o}, \tag{A.44}$$

so

$$E_n \leq o * P_{n,o} + (1 - P_{n,o}) * \sum_{k=o+1}^{\infty} kP_{1,k|\neg o}. \tag{A.45}$$

The sum in the above equation is constant and independent of n thus, we can write,

$$E_n \leq o * P_{n,o} + (1 - P_{n,o}) * C_3 \tag{A.46}$$

where $C_3 = \sum_{k=o+1}^{\infty} kP_{1,k|\neg o}$. Substituting Equation A.42 for $P_{n,o}$ allows us to rewrite this as

$$E_n \leq C_1 + C_2 * (1 - P_{1,o})^n \tag{A.47}$$

where $C_1 = o$ and $C_2 = C_3 - o$.

In general, for $0 \leq x < 1$, $(1 - x)^n$ is $\Omega(1/n)$. Thus, E_n decreases to the minimum required learning time at a rate that is $\Omega(1/n)$.

Next, recall that E_n is the expected time required for an agent to solve the task initially. To learn the actions along an optimal path may require the agents to solve the task multiple times, where each solution involves subsequently shorter and shorter random walks. In general, the expected solution times for these additional trials have a form similar to Equation A.47. Thus, in general, the expected solution time decreases towards o at a rate that is $\Omega(1/n)$.
□

Theorem 11: *If a naive 1-step Q-learning agent using LBW and a skilled (optimal) role model solve identical tasks in parallel and if the naive agent aborts and restarts the task after failing to solve it in n_q steps, then an upper bound on the* *time needed by the naive agent to first solve the task (and learn the actions along the optimal path) is given by*

$$\left\lceil \frac{i^2}{n_q} \right\rceil n_q + i. \qquad \text{(A.48)}$$

Proof:
This follows since after i^2 steps the naive agent will have watched the role model solve the task at least i times, the number of times required for 1-step Q-learning to propagate credit along the optimal path. Thus, after i^2 steps the naive agent will know how to solve the task from the start state. However, because it may be in the middle of a trial it may perform a total of $\left\lceil \frac{i^2}{n_q} \right\rceil n_q$ steps before quitting for the last time and beginning the final trial. Once the final trial begins, the agent will solve the task in i steps. Thus, the total number of steps required is at most

$$\left\lceil \frac{i^2}{n_q} \right\rceil n_q + i. \qquad \text{(A.49)}$$

□